CodeWarrior®

Error Reference

Because of last-minute changes to CodeWarrior, some of the information in this manual may be inaccurate. Please read the Release Notes on the CodeWarrior CD for the latest up-to-date information.

Revised: 980829-JDR

Metrowerks CodeWarrior copyright ©1993–1998 by Metrowerks Inc. and its licensors. All rights reserved.

Documentation stored on the compact disk(s) may be printed by licensee for personal use. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from Metrowerks Inc.

Metrowerks, the Metrowerks logo, CodeWarrior, and Software at Work are registered trademarks of Metrowerks Inc. PowerPlant and PowerPlant Constructor are trademarks of Metrowerks Inc.

All other trademarks and registered trademarks are the property of their respective owners.

ALL SOFTWARE AND DOCUMENTATION ON THE COMPACT DISK(S) ARE SUBJECT TO THE LICENSE AGREEMENT IN THE CD BOOKLET.

How to Contact Metrowerks:

U.S.A. and international	Metrowerks Corporation 9801 Metric, Suite 100 Austin, TX 78758 U.S.A.
Canada	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
Ordering	Voice: (800) 377–5416 Fax: (512) 873–4901
World Wide Web	`http://www.metrowerks.com`
Registration information	`register@metrowerks.com`
Technical support	`support@metrowerks.com`
Sales, marketing, & licensing	`sales@metrowerks.com`
CompuServe	goto `Metrowerks`

Table of Contents

Table of Contents 3

1 Introduction 5

Metrowerks Year 2000 Compliance 5
Overview of the Error Reference 5
Conventions Used in This Manual 6
Settings Affect Errors 7

2 C/C++ Compiler Error Messages 9

C/C++ Compiler Errors 9
Symbol Names (C/C++) 9
Punctuation (C/C++) 11
A to C (C/C++) 17
D to F (C/C++) 25
G to I (C/C++) 32
J to L (C/C++) 63
M to O (C/C++) 65
P to R (C/C++) 68
S to T (C/C++) 74
U to Z (C/C++) 82

3 Pascal Compiler Error Messages 87

Pascal Compiler Errors 87
Symbol Names (Pascal) 87
Punctuation (Pascal) 89
A to C (Pascal) 95
D to F (Pascal) 111
G to I (Pascal) 118
J to L (Pascal) 134
M to O (Pascal) 135
P to R (Pascal) 150
S to T (Pascal) 157
U to Z (Pascal) 164

4 Java Error Messages 175

Java Compiler Errors 175
Symbol Names (Java) 175
Punctuation Marks (Java) 180
A to B (Java) 180
C (Java) 191
D to F (Java) 211
G to I (Java) 217
J to N (Java) 232
O to R (Java) 247
S to U (Java) 253
V to Z (Java) 268

5 Linker Error Messages 275

Linker Errors 275
- Symbol Names (Linker) 275
- A to C (Linker) 279
- D to F (Linker) 285
- G to I (Linker) 288
- J to L (Linker) 291
- M to O (Linker) 292
- P to T (Linker) 295
- U to Z (Linker) 299

1 Introduction

Metrowerks Year 2000 Compliance

The Products provided by Metrowerks under the License agreement process dates only to the extent that the Products use date data provided by the host or target operating system for date representations used in internal processes, such as file modifications. Any Year 2000 Compliance issues resulting from the operation of the Products are therefore necessarily subject to the Year 2000 Compliance of the relevant host or target operating system. Metrowerks directs you to the relevant statements of Microsoft Corporation, Sun Microsystems, Inc., Apple Computer, Inc., and other host or target operating systems relating to the Year 2000 Compliance of their operating systems. Except as expressly described above, the Products, in themselves, do not process date data and therefore do not implicate Year 2000 Compliance issues.

For additional information, visit

```
http://www.metrowerks.com/about/y2k.html
```

Overview of the Error Reference

This manual lists the errors you may encounter from the CodeWarrior compilers and linkers.

When you compile and link code, CodeWarrior may discover problems. If there is a problem, the compiler or linker posts an error in the Message window. This manual discusses each error, what it means, and in many cases provides you with suggestions for correcting the error.

In each chapter, the errors are arranged in sections according to the first character in the message. Errors that begin with a symbol name

(such as the name of a variable or routine) come first. Errors that begin with a non-alphabetic character (such as punctuation) come next. After that, the errors are listed alphabetically.

The chapters in this manual describe the errors you may encounter from each compiler and linker. They are:

- "C/C++ Compiler Errors" on page 9
- "Pascal Compiler Errors" on page 87
- "Java Compiler Errors" on page 175
- "Linker Error Messages" on page 275

Conventions Used in This Manual

The following list describes the font conventions and structure used in this reference manual.

Error Message *variable name*

This section following the error message explains the nature of the error and its possible causes. A name in italics indicates an unknown variable name or type that will be filled in by the compiler when the error is generated.

')' expected

The compiler did not find a right parenthesis where it expected to find one.

NOTE: An example of how an error or warning message would appear.

Listing 1.1 sample source code

```
In some cases, sample source code is provided that demonstrates
the error message.
```

Fix: Some error messages include a suggestion about how the error could be fixed.

See Also Some error messages include a reference where more information can be found. This section often points out a Metrowerks CodeWarrior feature that can be used to detect or stop the error from being generated.

Settings Affect Errors

Language options not only affect the way the compiler translates the source code in your program files,. These settings also affect which source code is flagged as errors and which source code compiles. In many cases, the appearance of errors depends heavily on the selected language settings. Where applicable, the description of an error will include the name of the checkbox that, when selected, screens for a particular error.

NOTE: For more information on the panels which affect error recognition, consult the CodeWarrior *IDE User Guide*

To change the language options used by the compiler, first select ***Target*** **Settings** from the **Edit** menu. Once the Target Settings dialog box appears, select the appropriate language settings panel from the list on the left, and choose the language options.

2

C/C++ Compiler Error Messages

This chapter gives an alphabetical list of the compiler errors which may be encountered while using Metrowerks CodeWarrior compilers for the PowerPC-based, 68K-based Mac OS, Win32/x86 and Windows NT application code generators.

C/C++ Compiler Errors

In this list, errors with variable initial text (such as a class or function name) come first. Errors beginning with a non-alphabetic symbol character come next. After that, errors are listed alphabetically.

NOTE: The description of some C++ errors contains a reference to Margaret A Ellis and Bjarne Stroustrup's *The Annotated C++ Reference Manual*, Addison-Wesley, Reading, MA, 1990. These page references are indicated by the abbreviation ARM, followed by the page number and section. For example: ARM p.202, 10.1.1 Ambiguities.

Symbol Names (C/C++)

These are C/C++ compiler error messages that begin with a symbol name, the name of a variable or function.

Error 10177 ***class* is not a SOM class**

You are using one of the SOM pragmas within the declaration of a class that isn't a SOM class. The pragmas `SOMReleaseOrder`, `SOM-`

`ClassVersion`, `SOMMetaClass`, and `SOMCallStyle` may appear only within the declaration of the SOM class they apply to.

Fix Be sure the pragma appears within the declaration of the SOM class it applies to. Don't use these pragmas with classes that are not descended from SOMObject.

Error 10164 ***variable* could not be assigned to a register**

Undocumented at this time.

Error 10051 ***variable* is not a struct/union/class member**

An item referenced as a member/method of a struct, union, or class is not defined as being a member/method. For example, in Listing 2.1 below, `theColors.color` references a member, `var`, that does not belong to the `struct ColorValues`.

Listing 2.1 Not a struct/union/class member

```
typedef struct
{
  short  seq; // var is not defined in
  short  group;// this struct
} ColorValues;

ColorValuestheColors;
  theColors.color = 1;// error: see above
```

Fix Check the structure, union, or class in question. This error may be caused by a simple spelling mistake. In C++ this error often happens when a class member function or constructor's arguments do not match the prototype. If this is not the case, either change the item referenced or modify the structure, union, or class.

Error 10193 ***classname* is not an Objective-C class**

This error is generated because the class you attempted to use `classname` is not an Objective-C type class.

Punctuation (C/C++)

These are C/C++ compiler error messages that begin with punctuation marks.

Error 10126 ***func* hides inherited virtual function *func2***

You declared a non-virtual member function that hides a virtual function in a superclass. One function hides another if it has the same name but a different argument types. For example:

Listing 2.2 func hides inherited virtual function *func2*

```
class A {
  public:
    virtual void f(int);
    virtual void g(int);
};

class B: public A {
  public:
    void f(char);// WARNING: Hides A::f(int)
    virtual void g(int); // OK: Overrides A::g(int)
};
```

This warning appears only if you turned on the **Hidden virtual functions** option in the C/C++ Warnings settings panel.

Fix Turn off the **Hidden virtual functions** option or choose another name for one of the functions.

Also, ensure that all derived virtual functions have identical parameter lists as the base virtual function.

Listing 2.3 Function declaration hides inherited virtual function.

```
class X     { virtual void f(); };
class Y : X { void f(int); };      // Y::f() hides X::f()
```

Error 10116 **#if nesting overflow**

This error occurs when the number of nested `#if` processor directives exceeds the maximum number allowed.

Fix To fix this error, study the logic behind your nested `#if`s. There's probably a way of dividing the large nested `#if` into a series of smaller nests.

Error 10144 **#include nesting overflow**

This error occurs when the number of nested `#includes` processor directives exceeds the maximum number allowed.

Fix To fix this error, study the logic behind your nested `#includes`. There's probably a way of dividing the large nested `#includes` into a series of smaller nests.

Error 10015 **'(' expected**

The compiler did not find a left parenthesis where it expected to find one.

Fix Use the **Balance** command to balance all left and right parenthesis.

Error 10016 **')' expected**

The compiler did not find a right parenthesis where it expected to find one.

Fix Use the **Balance** command to balance all left and right parenthesis. This error may be caused by a syntax error in a previous statement.

To prevent this error while typing in source code, select the **Balance While Typing** checkbox in the Editor Settings preference panel. When selected, this preference sounds an alert if an un-matching right parenthesis is typed.

See Also For more on **Balance While Typing**, consult the *CodeWarrior IDE User's Guide*.

NOTE: This error may be caused by a syntax error or missing symbol in a previous statement.

Error 10131 **'<' expected**

The compiler did not find a left angle bracket where it expected to find one.

Fix This error may be caused by a syntax error or missing symbol in a previous statement.

NOTE: The Balance command does not check for angle brackets, '<' and '>'.

Error 10132 **'>' expected**

The compiler did not find a right angle bracket where it expected to find one. For example, Listing 2.4 gives an example of a missing right angle bracket.

Listing 2.4 **'>' expected**

```
template <class T> class Ca;
CA *aClass;                        //  error
```

Fix This error may be caused by a syntax error in a previous statement.

NOTE: The Balance While Typing does not check for angle brackets, '<' and '>'.

See Also For more on **Balance While Typing**, consult the *CodeWarrior User's Guide*.

Error 10017 **‘,’ expected**

The compiler did not find a comma where it expected to find one.

Fix This error may be caused by a previous syntax error. For example, the compiler expects to find a comma in the call to `GetMenu()` in the Listing 2.5 below. Actually, to fix this error, a parenthesis must be added to end the function call to `GetMenu()`.

Listing 2.5 Comma expected

```
gAppleMenu = GetMenu(APPLE_MENU_ID);
gFileMenu  = GetMenu(FILE_MENU_ID;
// the error compiler expected a comma because a right
parenthesis is missing after FILE_MENU_ID.
```

Error 10071 **‘:’ expected**

The compiler did not find a colon where it expected to find one. For example, in the `switch` statement in Listing 2.6, a colon is missing after `APPLE_MENU_ID`.

Listing 2.6 Colon expected

```
switch(theMenu)
{
  case APPLE_MENU_ID// error:  missing ':'

switch(theItem)
{
  case ABOUT_ITEM : // Correct
```

Fix If the error is not apparent, an error in a previous statement may be causing the problem. Correct all previous errors first and recompile.

Error 10024 **';' expected**

The compiler did not find a semicolon where it expected to find one. For example, in Listing 2.7 below, a semicolon is missing after the function call `WindowInit()`.

Listing 2.7 Semicolon expected

```
ToolBoxInit();
WindowInit()// ';' missing from this line
MenuBarInit();
```

Fix If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

'[' expected

The compiler did not find a left bracket where it expected to find one.

Fix If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

To prevent this error while typing in source code, select the **Balance While Typing** checkbox in the Editor Settings preference panel. When selected, this preference allows you to balance brackets as you type them.

See Also For more on **Balance While Typing**, consult the *CodeWarrior IDE User's Guide*.

Error 10026 **']' expected**

The compiler did not find a right bracket where it expected to find one.

Fix If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

To prevent this error while typing in source code, select the **Balance While Typing** checkbox in the Editor Settings preference panel. When selected, this preference allows you to balance brackets as you type them.

See Also For more on **Balance While Typing**, consult the *CodeWarrior IDE User's Guide*.

Error 10036 **'{' expected**

The compiler did not find a left brace where it expected to find one.

Fix Use the **Balance** command to balance all left and right braces. This error may be caused by a syntax error in a previous statement.

Error 10031 **'}' expected**

The compiler did not find a right brace where it expected to find one.

Fix Use the **Balance** command to balance all left and right braces. This error may be caused by a syntax error in a previous statement.

To prevent this error while typing in source code, select the **Balance While Typing** checkbox in the Editor Settings preference panel. When selected, this preference lets you balance braces as you type them.

Error 10157 **'&' reference member *var* is not initialized**

A reference member was not initialized. All reference types must be evaluated in the scope of the constructor. For example, Listing 2.8 below shows an un-initialized and a properly initialized reference.

Listing 2.8 Reference member not initialized

```
class caClass {
   private:
      int x;
   public
      const int &ref;
```

```
      caClass() {}  //  <-- no  initialization
 };
  // properly initialized reference
class caClass {
   private:
      int x;
   public:
      const int &ref;
      caClass():ref(x) {} //<-- now reference is initialized
 };
```

A to C (C/C++)

These are error messages that begin with *A*, *B*, or *C*.

Error 10089 **ambiguous access to class/struct/union member**

The compiler signals this error when a reference to a class, struct, or union member is ambiguous.

You may get this message when calling a function that is defined in both a virtual base class and another base class with the same parameters. The CodeWarrior C++ compiler is more strict about this situation. You must use the fully qualified form to define which function you want to use.

See Also ARM p.202, 10.1.1 Ambiguities.

Error 10220 **ambiguous access to name found '*symbol*' and '*symbol*'**

The compiler generates an error when it sees ambiguous access to a name. This usually occurs when the same name is used in multiple, legally accessible namespaces, as shown in Listing 2.9.

Listing 2.9 Ambiguous access to name

```
namespace A {
  int a;
}
```

```
long a;
namespace B {
  using namespace ::A;
  int x = a; //<<<What is this ‘a’? ERROR.
  }
```

Error 10100 **ambiguous access to overloaded function**

This error is displayed when an ambiguous reference is made to an overloaded function. References to overloaded functions must be unambiguous. InListing 2.10, `funcA()` is overloaded with a default argument. When it is called in the `main()` function, the compiler does not understand which member function to use.

NOTE: This error is also common using `double`/`float` or `short`/`long` arguments to overload a function.

See Also ARM p.307, 13 Overloading.

Listing 2.10 **Ambiguous access to overloaded function**

```
class aClass
{
    int x;
  public:
    int funcA( ) { x = 2; return x; }
    int funcA(int y = 3)  { x = y; return x; }
};

main()
{
  aClass obj;

  obj.funcA();
  return 0;
}
```

Error 10206 **ambiguous message selector used: *msg* also had: *msg***

This error is generated when there is ambiguity of which selector should be used.

Error 10247 **ambiguous use of partial specialization**

The compiler generates this error when more than one partial specialization matches the class type.

Error 10217 **assigning a non-int numeric value to an unprototyped function**

(Warning) This warning is activated by #pragma warn_largeargs on, or by passing -warn largeargs to the command-line compilers.

The compiler will emit a warning when passing a non-integer numeric value such as a float or a long long to an unprototyped function when the "require prototypes" option is off.

Error 10186 **assignment is not supported for SOM classes**

You cannot use a class descended from SOMObject in an assignment operation, since SOM classes do not support copy constructors. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide*, or the *SOMObjects Developer Toolkit* (IBM).

Error 10059 **branch out of range**

A branch destination in an assembly function is out of range, as in the branch call in Listing 2.11.

Listing 2.11 **Branch out of range**

```
bra.s 10000
```

Error 10062 **call of non-function**

An attempt was made to call a non-function. For example, Listing 2.12 below attempts to call the variable `i` as if it were a function.

Listing 2.12 Call of a non-function

```
main()
{

int i;
...
  i();// error: “i” is not a function
...
}
```

Fix Check the non-function call in question. This error may be caused by a spelling mistake that attempts to call a similarly named non-function instead of the desired function.

Error 10228 **cannot allocate initialized objects in the scratchpad**

(PlayStation only) Static initialization in the scratch pad is not supported. Call a routine to initialize it, as shown in Listing 2.13.

Listing 2.13 Cannot allocate initialized objects in the scratchpad

```
__declspec(scratchpad) int P=1234;// ERROR
__declspec(scratchpad) int P;     // OK
...
void InitScratchPad()
{
    P = 1234;
}
```

Error 10114 **cannot construct base class *aClass***

This error appears when the base class *aClass* has no ctor initializer or default constructor.

Error 10115 **cannot construct direct member *aClass***

This compiler gives this error when the direct member *aClass* has no ctor initializer or default constructor.

Error 10145 **cannot convert from *Type_A* to *Type_B***

The compiler generates this error message when a type conversion was attempted without proper conversion constructors, or with incompatible types. The code in Listing 2.14 attempts to convert a type `long *` to an `int *`.

Listing 2.14 Cannot convert Type_A to Type_B

```
main()
{
 int  *ptr;
 ptr = new long;  // <-- Error  wrong type
 return 0;
}
```

Error 10154 **cannot delete pointer to const**

The compiler generates this error message when it encounters an attempt to delete a pointer to a const value. For example, in Listing 2.15 below an attempt is made to delete a pointer to a `const` type.

Listing 2.15 Attempt to delete a pointer to a const

```
main()
{
 const int y = 3;
 int const *ptr = &y;
 delete ptr; // <-- Error here

 return 0;
}
```

Error 10155 **cannot destroy const object**

The compiler generates this error when an attempt to destroy a `const` object is encountered.

Error 10134 **cannot instantiate *obj***

The compiler generates this error when the template _*obj* cannot be instantiated because its definition is missing, such as the Listing 2.16 below.

Listing 2.16 Cannot instantiate

```
template <class T> class aClass;
template class aClass<int>;        //  error
```

Error 10137 **cannot pass const/volatile data object to non-const/volatile member function**

This compile error occurs when an attempt is made to pass a data object declared as a `const` to a member function that is not declared as `const`. The Listing 2.17 below generates this compiler error.

Listing 2.17 Cannot pass const data object to non-const member function

```
struct stType {
    void bar();             //  non-const member function
    void cbar() const;      //  const member function
};
...
    stType f;
    const stType cf;

    f.bar();    //  OK
    f.cbar();   //  OK
    cf.bar();   //  error
    cf.cbar();  //  OK
```

Error 10151 **cannot throw class with ambiguous base class *cBase***

The class in the throw point has an ambiguous base class.

Fix Declare a virtual base class or eliminate any ambiguities to resolve this error message.

Error 10073 **case constant defined more than once**

A constant used in a switch statement is already in use. For example, in Listing 2.18 below, the constant `ABOUT_ITEM` is used more than once.

Fix Remove one of the constant labels.

Listing 2.18 Case constant defined more than once

```
switch(theItem)
{
  case ABOUT_ITEM :
    Alert(ABOUT_ALRT, NIL);
    break;
  case ABOUT_ITEM :
    Alert(ABOUT_ALRT, NIL);
    break;
```

Error 10143 **'catch' expected**

Undocumented at this time.

Error 10156 **const member *aVar* is not initialized**

This error message is generated when the compiler encounters a const member that was not initialized correctly.

Fix The const member must be initialized at the time of the object's construction.

Error 10212 **category *Cat* redefined**

The compiler generated this error because you attempted to define a category `Cat` that was previously defined.

Error 10213 **category *Cat* is undefined**

The compiler generated this error because you attempted to use a category `Cat` that has not been defined.

Error 10196 **class *classname* redeclared**

This error is generated because you attempted to declare the class `classname` but it had previously been declared.

Error 10197 **class *classname* redefined**

This error is generated because you attempted to define a class `classname` but it had previously been defined.

Error 10104 **class has no default constructor**

This error occurs when the compiler cannot construct a class because it has no default constructor, as shown in Listing 2.19.

Listing 2.19 Class has no default constructor

```
struct stType {
  stType(int);
};

stType f; // error: no default constructor
```

Error 10147 **class type expected**

The compiler generates this error message when a class type was expected.

Error 10125 **'const' or '&' variable needs initializer**

You must initialize const or reference variables when you declare it. For example:

Listing 2.20 Const and reference variables need initializers

```
const int a = 1;// OK
const int b;  // ERROR: const variable
                  //      needs initializer

int c;
int &rc = c;  // OK
int &rd;  // ERROR: reference variable
                  //     needs initializer
```

Error 10159 **constness casted away**

This error message is generated when an attempt to cast a `const` to a `volatile` type is encountered.

D to F (C/C++)

These are C/C++ compiler error messages that begin with *D*, *E*, or *F*.

Error 10230 **data object *object* redefined**

The compiler generates an error when a data object is incorrectly redefined.

Listing 2.21 Data object *object* redefined

```
int a = 1;
int a = 2;//<<< ERROR: the variable name is reused incorrectly
```

Error 10046 **data type is incomplete**

The data type usually a class or structure is incomplete. “Incomplete class” errors usually happen when attempts are made to use classes that have been partially declared usually using “forward declarations”.

See Also “illegal use of incomplete struct/union/class” on page 57

Error 10022 **declaration syntax error**

The compiler encountered a syntax error while trying to resolve a declaration.

Fix Examine the declaration in question. If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

Error 10035 **declarator expected**

The compiler expected to find a declaration, but found something else instead.

Fix Check the declaration in question. If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

Error 10074 **default label defined more than once**

The compiler found more than one `default` label in the same `switch` statement. For example,Listing 2.22 below causes this error.

Listing 2.22 **More than one `default:`**

```
switch(...)
{
  default:;
  default:;// only one default in switch !
}
```

Fix Remove one of the `default` labels.

Error 10128 **derived function differs from virtual base function in return type only**

The compiler generates this error when the return type of the derived function differs from the return type of a virtual base function. The Listing 2.23 provides an example.

Listing 2.23 Derived function differs from virtual base in return type only

```
class aClass { virtual int f(); }
class bar : aClass {
  void f();        //  error
}
```

Error 10040 **division by 0**

When a constant expression tries to divide by zero or use modulo zero, a division by 0 error is signaled.

Error 10014 **end of line expected**

This error can occur in many circumstances, and may be the result of another error on a previous line of code. For example, if you turn on the **ANSI Keywords Only** option in C/C++ Language settings panel, this error occurs when text follows the `#endif` directive. The ANSI standard specifies that only a comment can follow an `#endif` directive. Listing 2.24 below shows another example where more tokens are expected on a line.

Listing 2.24 More tokens expected on a line

```
#define// error:  compiler expects more tokens
#if    //        on both lines.
```

Fix In the case of text rather than a comment following the `#endif` directive, deselect the **ANSI Keywords Only** checkbox in C/C++ Language settings panel, and recompile.

Error 10166 **exception specification list mismatch**

The exception specification lists for a function declaration and a function definition don't match, as shown in Listing 2.25.

Listing 2.25 exception specification list mismatch

```
void f() throw(int);
        // exception specification list mismatch
```

```
void f() throw(long)
{
}
```

exception handling does not work with 'direct destruction'

The compiler generates this error when you use C++'s exception handling and the **Enable C++ Exceptions** option in the C/C++ Language settings panel is off or the `direct_destruction` pragma is on.

Fix Turn on the **Enable C++ Exceptions** option in the C/C++ Language settings panel or turn off the pragma `direct_destruction`.

Error 10153 **exception handling option is disabled**

This error occurs when the Enable C++ Excpeptions option in the **C/C++ Language** settings panel is disabled, and you try to use EH (for example, `throw;`).

Error 10042 **expression syntax error**

The compiler generates this error when it encounters any kind of illegal expression syntax.

Fix If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

Error 10066 **function already has a stackframe**

This error occurs when the compiler finds more than one `fralloc` directive in an assembly function.

Fix Remove all but one of the `fralloc` directives from the assembly function.

Error 10149 **function call *func* does not match**

The compiler generates this error when a call to a function does not match the expected arguments. An attempt to initialize an object without a proper matching constructor also generates this message.

Fix Add a default constructor for your class. Also, check the previous defined class or structure, including the header file named prior to this error message, for a missing object list.

Listing 2.26 Function ‘func’ does not match

```
class A {
  public:
  A() {}
}  //<-- no semicolon, object list expected

main()
{
 A a;  // <-- error reported here
  return 0;
}
```

Error 10063 **function call does not match prototype**

A function call’s arguments do not correspond to the function’s prototype. Listing 2.27 shows that the call to `SetWaggle()` passes two arguments when the function prototype requires one.

Fix Match function call with function prototype. Select the function call and choose **Find Definition** from the **Search** menu to locate the function prototype definition.

Listing 2.27 Function call does not match prototype

```
long SetFoo(long foonum)
{
...
}
...
MyFoo = SetFoo(size, length);
  // error: two variables parameters in call to
  //      SetFoo, prototype has only one argument
```

Error 10141 **function defined 'inline' after being called**

This error message is generated when a function is declared inline after it has already been called. In Listing 2.28 below the function is used in the class before the function definition where it is declared inline.

Fix Declare the function prototype to be inline.

Listing 2.28 Function defined inline after being called

```
int func( int x );

 class cA {
    int i;
    public:
    cA() { i = func( 3); }
};

 inline int func( int x ) { return x + 1; }
```

Error 10067 **function has no initialized stackframe**

This error occurs when the compiler encounters an assembly function whose stack frame as not been allocated using the `fralloc` directive.

Fix Modify your assembly function so is that it uses `fralloc`.

Error 10079 **function has no prototype**

A function is defined without a preceding prototype. This error occurs if the **Require Function Prototypes** checkbox, in the C/C++ Language settings panel, is selected.

Fix Either define a preceding prototype for the function in question, or deselect the Require function prototypes checkbox.

See Also For more on the **Require Function Prototypes** option, consult the *C Compilers Reference*

Error 10069 **function nesting too complex**

This error occurs when the compiler encounters a function that contains too many nested `{}` blocks.

Fix To fix this error, study the function in question. There's probably a way to divide the function into a series of dependent functions.

Error 10184 **functions cannot return SOM classes**

A function cannot return a class which is descended from SOMObject, since SOM classes do not support copy constructors. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide*, or the *SOMObjects Developer Toolkit* (IBM).

Error 10185 **functions cannot have SOM class arguments**

A function cannot contain in its argument list any class which is descended from SOMObject, since SOM classes do not support copy constructors. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide*, or the *SOMObjects Developer Toolkit* (IBM).

Error 10227 **function result is a pointer/reference to an automatic variable**

The compiler generates an error when you use an automatic variable and reference it outside of its scope, as shown in Listing 2.29.

Fix Expand the scope or increase the persistence of the variable. For example, you could declare the variable as a pointer in the heap instead of declaring it as a local variable on the stack.

Listing 2.29 **Function result is a pointer/reference to an automatic variable**

```
char *foo()
{
  char c=0;
  return &c;
}
```

G to I (C/C++)

These are C/C++ compiler error messages that begin with *G*, *H*, or *I*.

Error 10189 **global SOM class objects are not supported**

If an object is a type descended from SOMObject, you cannot use it as a global variable. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide*, or the *SOMObjects Developer Toolkit* (IBM).

Error 10023 **identifier *name* redeclared**

Undocumented at this time.

Error 10150 **identifier *name* redeclared was declared as: *a_type'* now declared as: *'b_type***

The compiler issues this error when the source code attempts to re-define an identifier. For example, in Listing 2.30 below, the identifier `var` is declared as both a `long` and a `Point`.

NOTE: This error is often the result of re-using the name of a Macintosh declared variable or function.

Fix Change the name of one of the identifiers.

Listing 2.30 **Identifier var redeclared**

```
  long    var;
  Double  temp;
  Point   var;
           // var has already been declared above

// Using a Macintosh Toolbox function name
// as a class name.

  class Random { public: void get(); };
```

```
void Random::get() // <-- this produces an error
{ //... }
```

Error 10008 **identifier expected**

The compiler expected to find an identifier, but instead found another token. For example, in Listing 2.31 below , the compiler issues this error because `short` is placed where an identifier should be.

Fix If the error is not apparent, it is likely being caused by a missing symbol, such as a semicolon or comma, on a previous line. Correct all previous errors first and recompile.

Listing 2.31 **Identifier expected**

```
long Waggle;
longtemp, short; // short is placed where an
                 // identifier should be
PointmyPt;
```

Error 10087 **illegal #pragma**

The compiler found an unrecognized `#pragma` directive.

Fix A list of `#pragma`s recognized by Metrowerks C is listed in the *C Compilers Reference* Consult this list to make sure the `#pragma` you are using exists and is spelled correctly.

Error 10097 **illegal '&' reference**

This error is issued when the compiler encounters an illegal & reference, as in Listing 2.32.

Listing 2.32 **Illegal '&' reference**

```
int &&g; // error: cannot dereference an integer twice
```

Error 10231 illegal access to local variable from other function

The compiler generates this error when you illegally try to access a local variable from another function.

Listing 2.33 Illegal access to local variable from other function

```
void bar()
{
  int a=0;

  struct nest {
    void foo()
    {
      a=2;//<<< ERROR: ‘a’ is bar's local variable, which cannot
          //            be accessed from nest::foo
    }
  };
}
```

Error 10142 illegal constructor/destructor declaration

The compiler generates this error when an attempt to declare a constructor or destructor is done in an illegal manner. Listing 2.34 attempts to call the constructor after the object is already initialized.

Listing 2.34 Illegal constructor declaration

```
class cA {
  public:
     void cA() {}
};

cA A;

main() {
  A.cA();
  return 0;
}
```

Error 10160 **illegal const/volatile '&' reference initialization**

The compiler generates this error message when either a `const` or a `volatile` reference was improperly initialized.

Fix This is often the result of not initializing a reference during object construction.

Error 10082 **illegal data in precompiled header**

The compiler issued this error because the precompiled header contained improper data.

Fix Remove the data, from your precompiled header and precompile it again. Refer to the C Compilers Guide for information on illegal items in a precompiled header.

Error 10165 **illegal exception specification**

You used a exception specification where it isn't allowed, as shown in Listing 2.35.

Listing 2.35 **illegal exception specification**

```
typedef void (*f)() throw(int);
                    // cannot use spec in typedef
```

Error 10148 **illegal explicit conversion from *type_A* to *type_B***

The compiler issued this error when an explicit conversion of one type to an improper type is encountered. Listing 2.36 attempts to convert a pointer to a class into a `long`.

Listing 2.36 **Illegal explicit conversion**

```
class D { };

 main()
 {
    long x;
```

```
    D d;
    x = (long)d;
    return 0;
 }
```

Error 10102 **illegal ‘friend’ declaration**

The compiler issues this error when it encounters an illegal declaration. For example,Listing 2.37 shows `friend` being illegally declared.

Listing 2.37 Illegal ‘friend’ declaration

```
int i;
class aClass {
  friend i;// illegal 'friend' declaration
};
```

Error 10103 **illegal ‘inline’ function definition**

This error is given when a function that has already been referenced is defined as inline.

Error 10094 **illegal ‘operator’ declaration**

This error is displayed when the compiler finds an illegal operator declaration, as in Listing 2.38.

Listing 2.38 Illegal ‘operator’ declaration

```
int operator +(int,int,int);
```

Error 10140 **illegal *class::constructor(argument)* copy constructor**

This error appears if a class constructor function is declared with an argument of the same class type, as in Listing 2.39.

Listing 2.39 Illegal class::constructor(argument) copy constructor

```
class aClass {
  aClass(aClass);// error
  aClass(aClass&); // OK
};
```

Error 10101 illegal access/using declaration

This error occurs when you attempt to incorrectly declare access to a base class member from a derived class. You can adjust the access to base class member by using the base class member's qualified-name, in a public or protected part of a derived class declaration. Listing 2.40 demonstrates this error.

See Also ARM p. 244, 11.3 Access Declaration.

Listing 2.40 Illegal access declaration.

```
class B {
    private:
  int a
    public:
  int b;
};

class D : private B {
    public:
  B::a; // <-- Error illegal access declaration
  B::b; // legal allows B::b to be used in
            //an external function
  int x;
  void fx() { b = x; }
}Derived;

inline void fx() { Derived.b = 3; }
```

Error 10093 **illegal access qualifier**

This error is signaled when an illegal qualification is encountered. In Listing 2.41, this code is illegal because `foo` doesn't exist.

See Also ARM p. 112.

Listing 2.41 Illegal access qualifier

```
struct stType { enum efoo { nfoo } mfoo; };
...
  foo::xfoo; // error: illegal qualified
```

Error 10088 **illegal access to protected/private member**

This error is signaled when a function attempts to access a private member. For example, in the Listing 2.42 `priv` is declared as `private`. Function `func()` attempts to access this member.

Listing 2.42 Illegal access to private member

```
class aClass { private: int priv; }
func() {
  aClass x;
  x.priv=0;
      //func() cannot access private member priv
}
```

Error 10056 **illegal addressing mode**

An assembly-language instruction attempts to use an addressing mode that is not possible with this instruction, as in Listing 2.43.

Listing 2.43 Illegal addressing mode

```
moveq  d0,d1
```

Error 10010 **illegal argument list**

This error appears when the compiler finds an illegal macro argument list, as in Listing 2.44.

Listing 2.44 Illegal argument list

```
#define macro(arg,,)
```

Error 10030 **illegal array definition**

The compiler gives this error when it encounters an array defined with a negative or zero subscript (also illegal array base type). Making the last member of a struct an empty array is a non-ANSI language extension that is not supported by Metrowerks CodeWarrior C/C++ as demonstrated in Listing 2.45.

Fix As a work around for the source in Listing 2.45 , the code should change to what's shown in Listing 2.46.

Listing 2.45 Illegal array definition

```
typedef struct {
  short howMany;
  Data *dataBase[];// error: non-ANSI extension
} DataBase
```

NOTE: Remember to change your allocation routines so that they allocate the right size for these structs. For example, in Listing 2.46 the proper allocation routine would be `sizeof(DataBase) - (nb_elements -1) * sizeof(Data).`

Listing 2.46 Fix for illegal array definition

```
typedef struct {
  short howMany;
```

```
  Data *dataBase[1]; // OK: now ANSI compliant
} DataBase
```

Error 10080 **illegal assignment to constant**

This error is issued by the compiler when an expression attempts to assign a value to a constant, as in Listing 2.47.

Listing 2.47 Illegal assignment to constant

```
const int i=5;
...
i=10; // cannot assign to a const
```

Fix Check the assignment in question. This error may be caused by a spelling mistake that attempts to assign a value to a similarly named constant instead of the desired variable.

Error 10039 **illegal bitfield declaration**

The compiler issues this error when a bitfield of size zero is declared. This error is also issued if too many bits are requested, as in the example below.

Fix Check the bitfield declaration in question to make sure that a size of zero is not declared, and that too many bits are not requested.

Listing 2.48 Illegal bitfield declaration

```
long err:33;
```

Error 10001 **illegal character constant**

This error is signaled when an attempt is made to assign an illegal character constant. Listing 2.49 contains three examples of illegal character constants.

Fix Examine the character constant to make sure it is assigned a legal character.

Listing 2.49 Illegal character constant

```
  char FooCh;
...
...

  FooCh = ''; // each of these assignments
  FooCh = '\x'; // contain illegal character
  FooCh = 'ddjdjdj';// constants.
```

Error 10229 **illegal class member access**

This error occurs when you try access a class member that doesn't exist.

Error 10025 **illegal constant expression**

This error is issued when the compiler encounters a constant expression that contains an illegal value or operator.

Fix Examine and correct the constant expression in question. If this error is not apparent, it is likely being caused by a previous error. Correct all previous errors and recompile.

Error 10113 **illegal ctor initializer**

This error is signaled when the compiler encounters an illegal ctor initializer. For example, the ctor initializer in Listing 2.50 is illegal because there's no `i` member of the `aClass`.

Fix If you are having problems with constructors of a sub-class, you are probably not naming the parent class explicitly, such as in Listing 2.51.

Listing 2.50 Illegal ctor initializer

```
class aClass {
  aClass(): i(12) {} // error: illegal ctor
                     //  (no i member)
};
```

`CButton` is a subclass of `Clickable` which takes a type `Display-System*` as its argument. In CodeWarrior C++ when making a constructor which passes parameters to the parent class, you need to name the parent class explicitly. You must use the following template:

```
BLAH::BLAH(parameter1, parameter2) :
    PARENT_OF_BLAH(parameter2)
```

Listing 2.51 Illegal ctor initializer, explicit parent class

```
CButton::CButton(long resourceBase,
  DisplaySystem* displaySystem) : (displaySystem)
```

Error 10057 illegal data size

The compiler issues this error when a line in an assembly function contains an illegal data size, as in Listing 2.52.

Listing 2.52 Illegal data size

```
move.Z #0,d9
```

Error 10106 illegal default argument(s)

This error is given when the compiler finds a function which contains one or more illegal arguments, as in the function prototype of Listing 2.53.

Listing 2.53 Illegal default argument

```
int func(int x=1, int z);
```

Error 10117 illegal empty declaration

The compiler gives this error when a declaration is missing an identifier, as in the declaration in Listing 2.54.

Listing 2.54 Illegal empty declaration

```
int ;
```

Error 10139 illegal explicit template instantiation

The compiler generates this error whenever it encounters an illegal explicit template instantiation. Listing 2.55 provides an example where the function `f()` was not properly declared as a template function.

Listing 2.55 Illegal explicit template instantiation

```
//template <class T>
void f();
template void f<int>();     //  error
```

Error 10236 illegal explicit template specification

This error occurs when there is something wrong with your explicit template specialization. For example,

```
template <> int a;
```

Error 10028 illegal function definition

This error is signaled whenever the compiler encounters an illegally defined function.

Fix If the error is not apparent, it is likely being caused by a previous error. Correct all previous errors first and recompile.

Error 10098 illegal function overloading

A common cause for this error is the declaration of functions with the same name and identical arguments, but different return types.

See Also ARM p.307, 13 Overloading.

Error 10029 **illegal function return type**

This error is given when the compiler finds a function that returns an array or function. A function cannot return an array or function. A function can only return a pointer to an array or function.

Fix Modify your code so that the function in question returns a pointer to the array or function.

Error 10121 **illegal implicit const pointer conversion**

(Warning Message) The compiler issues this warning when you convert a `const` pointer into a variable, as in Listing 2.56.

Listing 2.56 **Illegal implicit const pointer conversion**

```
void func(const char *cptr)
{
  char *ptr=cptr;
      // illegal implicit const pointer conversion
}
```

Error 10110 **illegal implicit conversion from *Type_A* to *Type_B***

This error is signaled when the compiler encounters an illegal implicit conversion as in Listing 2.57.

NOTE: ANSI C++ differs from ANSI C in the treatment of `void*`. ANSI C allows an implicit conversion from a pointer to void to a pointer to another object type (but not to a pointer to function type--see Section 5.4); in C++ a `void*` cannot be assigned to an object of any type other than `void*` without an explicit cast. Thus, Listing 2.57 is legal ANSI C, but is not accepted in C++:

See Also ANSI Draft Standard Section 5.4 "Pointer Conversions," ANSI C Standard 3.2.2.3

Listing 2.57 Illegal implicit conversion

```
void f(char *cptr, void *vptr)
{
  cptr = vptr;
                  // Illegal in C++, legal in C
  char *ptr=(void *)0;
                  // Illegal in C, legal in C++
  // . . .
}
```

Error 10118 **illegal implicit enum conversion from *Type_A* to *Type_B***

The compiler gives this message when an illegal implicit conversion, involving an enum, is encountered. If the source code is C++, the compiler gives this message as an error. If the source code is C and the **Extended Error Checking** option in the C/C++ Warnings settings panel is on, the compiler gives this message as a warning. An example is shown below in Listing 2.58.

Listing 2.58 Illegal implicit enum conversion

```
enum ff { foo };
enum ff x = 0;
//error:illegal implicit enum conversion
```

Error 10232 **illegal implicit member pointer conversion**

This error occurs when a member function is incorrectly initialized.

Listing 2.59 Illegal implicit member pointer conversion

```
struct X {
    void foo()
    {
      void (X::*f)() = foo;    // ERROR
      void (X::*g)() = &X::foo;// OK
    }
  };
```

Error 10075 **illegal initialization**

This error occurs when a variable, or other data type, is illegally initialized within a function.

Error 10053 **illegal instruction for this processor**

The compiler issues this error when an assembly-language instruction is found that does not exist for the 68000 family of microprocessors.

Error 10112 **illegal jump past initializer**

This error is signaled when a transfer is made into a block that bypasses initializers. In certain cases, it is illegal to jump past explicit or implicit initializers. Generally this error occurs whenever there is a section of code that can be jumped past in the same scope. Typically in `switch` or `goto` statements as in Listing 2.60.

NOTE: It's possible for the definition of p to be skipped over within the scope it's in (the switch statement). The fix to this is to either define p outside of the switch, or make a new scope. If you have any `goto` statements in your function, you'll get this error for any variables that are defined after a `goto`. The solution is the same: either define all variables before the `goto`, or introduce a new scope.

See Also ARM p. 87, 6.4.2 The Switch Statement and p.91, 6.7 Declaration Statement.

Listing 2.60 **Illegal jump past initializer**

```
switch (i) {
  int v1 = 2;  // error
   case 1:
     short v2 = 3;
   case 2:
```

```
      if( v2 == 7 ) {}  // error
  }
```

Error 10204 **illegal message receiver**

This message is given when you try to send a message to a non-ObjC object.

Error 10221 **illegal namespace**

This error is generated when you use a non-namespace name as a namespace.

Listing 2.61 **Illegal namespace**

```
int a;
namespace a {//<<< ERROR
    int b;
}
```

Error 10223 **illegal name overloading**

The compiler generates an error when you attempt to perform an illegal overload, as shown in Listing 2.62.

Listing 2.62 **illegal name overloading**

```
int b;
enum { b };//<<< ERROR
```

Error 10045 **illegal operand**

This error is signaled when an operator is applied to a non-compatible operand.

Fix Try type-casting the operand to a compatible type.

Error 10054 **illegal operands for this processor**

This error is issued when the compiler encounters an assembly-language instruction that refers to operands that do no exist for the 68K family of microprocessors.

Error 10044 **illegal operation**

An operator, such as == or + was illegally applied to a struct or union. This error is also signaled when an operator is not defined for a data type.

Error 10105 **illegal operator**

This error is signaled when the compiler encounters an illegal operator, as in Listing 2.63.

Listing 2.63 **Illegal operator**

```
int operator .(int i);
```

Error 10099 **illegal operator overloading**

A common cause for this error is trying to overload an operator that cannot be overloaded, or trying to overload a preprocessor directive.

See Also ARM p.329, 13.4 Overloaded Operators

Error 10245 **illegal partial specialization**

The compiler gives this error when you have an error in a partial class specialization.

Error 10246 **illegal partial specialization argument list**

The compiler gives this error when you have an error in the partial class specialization argument list.

Error 10124 **illegal precompiled header compiler flags or target**

The compiler gives this error when a precompiled header file uses the wrong compiler target. For example, you get this error if you ar compiling code for a Power Mac OS computer with a precompiled header that has a first line of `#include <MacHeaders68k>`.

Fix Check your pre-compiled header or `.pch` file for flags or data of a different CPU type than your current target. Also check the prefix file in the C/C++ Language settings panel.

Error 10123 **illegal precompiled header version**

The compiler gives this error when a precompiled header file is old or defective.

Fix Possible fixes include:

- You may need to precompile your project's precompiled header again.
- If you are using a prefix file provided by Metrowerks, you should re-build the precompiled header projects (these include `MacHeaders.mcp`, `Win32Headers.mcp`, and `MSL-Headers.mcp)`.

Error 10058 **illegal register list**

This error occurs when the compiler encounters an illegal register list in an assembly function. An example is shown in Listing 2.64.

Listing 2.64 **Illegal register list**

```
movem.ld0-d0,—(sp)
```

Error 10216 **illegal return value in void/constructor/destructor function**

This error message is generated if you attempt to return a value from a void function, or a constructor or destructor which by design may not return a value.

Fix Remove the illegal return.

Listing 2.65 Example of illegal return value.

```
void foo() { return 1; }
```

Error 10171 **illegal SOM function overload *operator***

You cannot overload member functions in a SOM class. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide*, or the *SOMObjects Developer Toolkit* (IBM).

Error 10174 **illegal SOM function parameters or return type**

You cannot use `long double` parameters or return type in member functions for SOM classes. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide,* or the *SOMObjects Developer Toolkit* (IBM).

Fix Rewrite the function using a different parameter or return type.

Error 10078 **illegal storage class**

The compile issues this error when an illegal storage class is used.

If the compiler points to a static member, it may mean that you have defined a member as `static`, ARM (p. 179) says "A data or function member of a class may be declared static in the class declaration" — but not in the definition.

Likewise, you may not declare variables as `auto` in global scope as in Listing 2.66.

See Also ARM , p.179.

Listing 2.66 Illegal storage class

```
auto int x;//error: auto is not allowed in global scope
```

Error 10002 **illegal string constant**

This error is displayed when a string constant is encountered that does not terminate before the end of a line.

Fix Terminate the string before the end of a line. If this error is not apparent, it is likely being caused by a previous error. Correct all previous errors and recompile.

Error 10032 **illegal struct/union/enum/class definition**

The compiler issues this error when an illegal struct, union, enum, or class definition is encountered.

Fix Examine the illegal struct, union, enum, or class definition to check for any syntax errors.

Error 10133 **illegal template argument(s)**

The compiler gives this error when it finds a template which contains one or more illegal arguments, as shown in Listing 2.67.

Fix This error is often the result of the omission of a space in nested templates creating a right shift operator.

Listing 2.67 Illegal template argument

```
map<double, double, less<double>> aMap;
    // illegal argument
map<double, double, less<double> > aMap;

template <class T> class aClass;
aClass<int,int> *aClassptr; // illegal argument
```

Error 10130 **illegal template declaration**

The compiler gives this error when a malformed template declaration is encountered, such as the template declaration in Listing 2.68.

Listing 2.68 Illegal template declaration

```
template <class T> T i;         //  error
```

Error 10006 illegal token

The compiler issues this error when an illegal preprocessor token is found. For example, the @ symbol in Listing 2.69 is an illegal token.

Fix Remove the illegal token. If the illegal token is not apparent, the error may be caused by a previous syntax error. Fix all previous errors and recompile.

Listing 2.69 Illegal token

```
if( @ )
```

Error 10047 illegal type

The compiler generates this error message when an illegal type is encountered as in Listing 2.70.

Listing 2.70 Illegal type

```
static void func(int i)
{
 delete i; // <-- illegal type
}
```

Error 10065 illegal type cast

This error occurs when the code attempts to typecast data to an incompatible data type.

Error 10077 illegal type qualifier(s)

The compiler issues this error when an illegal type qualifier, for this type in this scope, is encountered. For example, the double `const`

qualifier in Listing 2.71 below will produce an illegal type qualifier error.

Fix Remove the illegal type qualifier. If this error is not apparent, it may be caused by a previous error. Correct all previous errors and re-compile.

Listing 2.71 Illegal type qualifier

```
const const int x;// double const
```

Error 10214 illegal use of *spec*

This error is generated when you use a keyword that is not in the correct syntax. This error is mostly used for qualifiers and specifers.

Listing 2.72 Example of illegal use of spec:

```
inline int k;       // illegal use of 'inline'
const int const cc; // illegal use of 'const'
```

Error 10242 illegal use of alloca() in function argument

(68K compiler only) The 68K version of `alloca()` cannot be used in function argument expressions, as shown in the example below.

```
#include <alloca.h>

extern void foo(void*,int);

static void bar()
{
   void *ptr;

   foo(alloca(100),1);   // error

   ptr=alloca(100);
   foo(ptr,1);           // OK
}
```

Error 10239 **illegal use of asm inline function**

This error is generated when you try to use entries or PC-relative data in inline assembly functions. An example of this error is shown in Listing 2.73.

Listing 2.73 Illegal use of asm inline function

```
extern void e();
inline asm int GetD7()
{
      move.l d7,d0
      entry  e
}
```

Error 10240 **illegal use of C++ feature in EC++**

This error is generated when you try to use a C++ feature that is not available in the EC++ language subset (for example, templates, namespaces, or multiple inheritance).

Error 10222 **illegal use of namespace name**

This error is generated when you use a namespace name as a non-namespace.

Listing 2.74 Illegal use of namespace name

```
namespace a {
      int b;
}
int g = a++;//<<< ERROR
```

Error 10208 **illegal use of Objective-C object**

This error is generated when you try to use an ObjC class in an unspupported way. For example if you pass the class by value, declare/define a ObjC class object.

Error 10178 **illegal use of #pragma outside of SOM class definition**

You can use four SOM pragmas only within the definition of the SOM class that the apply to: `SOMReleaseOrder`,`SOMClass-Version`, `SOMMetaClass`, `SOMCallStyle`. For more information on SOM objects, see the Metrowerks manual*C Compilers Guide*, or the *SOMObjects Developer Toolkit* (IBM).

Error 10119 **illegal use of #pragma parameter**

This message is displayed when a previous `#pragma` parameter does not match a function. For example, `func()` in Listing 2.75 below does not match the function `func1`.

NOTE: In addition, If the Illegal Pragmas option is selected in the C/C++ Warnings settings panel, undefined `#pragmas` are marked as warnings.

See Also For more on the `#pragmas` supported by Metrowerks C/C++, consult the *C, C++, and Assembler Language Reference*, Chapter 5, "Pragmas & Predefined Symbols." For more on the C/C++ Warnings settings panel, *C, C++, and Assembler Language Reference*, Chapter 2, "C and C++ Language Notes."

Listing 2.75 **Illegal usage of #pragma parameter**

```
#pragma parameter __A0 func
char *func1()
{
  // ...
}
```

Error 10092 **illegal use of 'HandleObject'**

The compiler gives this error when an illegal usage of a class that is derived from the `HandleObject` class is encountered.

Error 10202 **illegal use of 'self'**

This error is given when the keyword `self` is used incorrectly.

Error 10203 **illegal use of 'super'**

This error is given when the keyword `super` is used incorrectly.

Error 10090 **illegal use of ‘this’**

This error is signaled when the compiler encounters C++ code that uses `this` in a non-member function.

Error 10027 **illegal use of ‘void’**

The compiler gives this error when an operator is incorrectly applied to a void type, or a variable is declared as a `void` type, as in Listing 2.76.

Listing 2.76 **Illegal use of ‘void’**

```
int  myInt;
long myNumber;
void myFoo;
//error: a variable cannot be declared as void
```

Error 10095 **illegal use of abstract class *(aClass)***

The compiler gives this error when you attempt to instantiate from an abstract class. An abstract class is defined with at least one pure virtual method. A pure virtual method is declared as

```
// pure virtual method
virtual type MethodName ( arguments ) = 0;
```

An abstract class requires you to make a subclass that provides methods to replace any pure virtual methods.

Fix Abstract classes must be subclassed before being instantiated. Define a non-abstract subclass and derive your object from that subclass.

See Also ARM p. 214, 10.3 Abstract Classes.

Error 10084 **illegal use of direct parameters**

This error is issued when a function, or another expression, references a direct parameter that is not supported. For example, the example below attempts to use `__X`. This is illegal because a direct parameter must be either `__D0` to `__D2`, `__A0`, `__A1`, or `__FP0` to `__FP3`.

Error 10037 **illegal use of incomplete struct/union/class**

This error is signaled when an incomplete struct, union, or class is used illegally. For example, in Listing 2.77, an attempt is made to create an incomplete object.

Fix To avoid the errors, you can try to include `bar.h` so you get the full class declaration. Sometimes this does not work because of circular references in the include files. Another option is to avoid inline (include file) references to member variables or methods of these partial classes. Don't inline the offending function. Put it in a separate implementation file that includes both `foo.h` and `bar.h`.

Listing 2.77 Illegal use of incomplete struct

```
struct A x;  //cannot create an incomplete object
```

This error often happens when you attempt to use classes that have been partially declared, usually using forward declarations as follows:

Listing 2.78 Using a class that has been partially declared

```
// foo.h
class Bar; // empty forward declaration

class aClass {
public:
  // trouble coming up...
  void DoIt(int x) { mBar->DoStuff(x); }
```

```
        ...
private:
  Bar* mBar;// you declare it
};

bar.hclass Bar {// actual declaration
                    // of the class
  public:
       void DoStuff(int x);
        ...
};
```

Error 10076 **illegal use of inline function**

This error is signaled when an inline function is used illegally. For example, in Listing 2.79 an attempt is made to take the address of an inline function.

Listing 2.79 **Illegal use of inline function**

```
pascal Handle NewHandle(Size byteCount) = 0xA122;
...
&NewHandle;// error: cannot take address of
               //       inline function
```

Error 10070 **illegal use of keyword**

This error occurs when a keyword is used illegally. In some cases, this error is caused by a previous syntax error or missing symbol. For example, in Listing 2.80, the illegal use of keyword error is caused by a missing colon.

Fix Fix all previous error messages. If error still persists, verify that you are using the keyword correctly as in Listing 2.81

Listing 2.80 **Illegal use of keyword**

```
  switch(theMenu)
  {
```

```
    case APPLE_MENU_ID// error:  missing ':'
      switch(theItem)
      {
        case ABOUT_ITEM :
        ...
        ...
      }
    break;// illegal use of keyword error here
              // caused by above missing colon
```

Listing 2.81 Illegal use of direct parameters

```
int func(int x:__X) {}
    // error: direct parameter __X not recognized
```

Error 10122 **illegal use of non-static member**

This error is signaled when an attempt is made to access a non-static member without having an object of that class, as in Listing 2.82.

Listing 2.82 Illegal usage of non-static member function

```
struct stType {
  stTypef();
};
// ...
  stTypef();
      // error: cannot call without a stType object
```

Error 10081 **illegal use of precompiled header**

This error is given when the compiler encounters a precompiled header file included illegally. A precompiled header file is used illegally when more than one precompiled header file is `#included` in the source code file (as in Listing 2.83).

Fix Check all the precompiled header files included with your project. If the error is not apparent, check the header specified in the **Prefix File** field in the C/C++ Language settings panel.

Listing 2.83 Illegal usage of precompiled header: too many

```
#include <MacHeaders>
#include <MacHeaders>
      // error: only one precompiled header
      //    file allowed
```

Fix This error often occurs when the precompiled header file has already been `#included` in the **Prefix File** field in the C/C++ Language settings panel. Or, when a precompiled header is `#included` after a function, variable, or type declaration, as in Listing 2.84.

Listing 2.84 Illegal usage of precompiled header: declaration

```
long l;
#include <MacHeaders>
        // error: precompiled header included
        //      following declaration.
```

Error 10096 **illegal use of pure function**

The compiler generated this message when it encountered an improper usage of a pure virtual function. A pure virtual function has no definition. For example in Listing 2.85 the constructor attempts to call the pure function `myFun()`.

See Also ARM, p. 214, 10.3, Abstract Classes.

Listing 2.85 Illegal use of pure function

```
class pure {
  public:
  virtual int myFun() = 0;
  pure() { myFun(); }
};
```

Error 10064 **illegal use of register variable**

This error occurs when a register is used illegally. For example, in Listing 2.86, an attempt is made to take the address of a register variable.

Listing 2.86 Illegal use of register variable

```
register int i;
f(&i);  // error: cannot take address of i
```

Error 10241 **illegal use template argument dependent type 'T::%u'**

The compiler gives this error when a template dependent type cannot be resolved. An example of this error is shown in Listing 2.87.

Listing 2.87 Illegal use template argument dependent type 'T::%u'

```
template <class T> int foo(typename T::x arg);
int i = foo<int>(1);
// error: illegal use template argument dependent type 'T::x'
```

Error 10218 **implicit arithmetic conversion from *Type_A* to *Type_B***

This warning occurs when you implicitly convert a big arithmetic type to a smaller (this has to be enabled in the Warnings prefs panel). For example,

```
long l;
short s;
...
 s=l; // <<<error
```

Error 10250 **implicit 'int' is no longer supported in C++**

This error occurs when you try to use an implicit `int` in C++ code. This option is no longer supported in C++.

```
  main()  // <<< must be changed to: int main()
```

Error 10161 **inconsistent linkage: ‘extern’ object redeclared as ‘static’**

The compiler will generate this error message in C++ if you try to re-declare or define an extern object as static. This is shown in Listing 2.88.

Listing 2.88 Inconsistent linkage: ‘extern’ object redeclared as ‘static’

```
void f();
static void f(); // <<< ERROR
```

Error 10244 **inconsistent use of ‘class’ and ‘struct’ keywords**

The compiler generates an error when you inconsistently use the keywords class and struct.

In the example below, `X` is first declared as a `class`, and then it’s inconsistently declared as a `struct`.

```
#pragma warn_structclass on
class X;
struct X { int a; }; // warning: inconsistent use of
                     // ‘class’ and ‘struct’ keywords
```

Error 10243 **inline function call *function name* not inlined**

You get this error when a function is not inlined (for example, when the inline level is not large enough).

Error 10224 **instance variable list does not match @interface**

The compiler generates an error when the instance variable list declared in the interface does not match what is defined in the implementation.

Listing 2.89 Instance variable list does not match @interface

```
@interface A
{
    int a;
```

```
}
@end
@implementation A
{
  long b; //<<< ERROR: inconsistent with declaration
}
@end
```

Error 10179 **introduced method *method* is not specified in release order list**

If you use the `SOMReleaseOrder` pragma for a SOM class, the pragma must list all the new methods that the class declares (but not the methods it overrides). For more information on SOM objects, see the Metrowerks manual *C Compilers Guide*, or the *SOMObjects Developer Toolkit* (IBM).

Fix There are two ways to fix this problem:

- Include the method in the `SOMReleaseOrder` pragma's list.
- Remove the `SOMReleaseOrder` pragma. The compiler assumes the release order is the same as the order in which the functions appear in the class declaration. However, when you release a version of the class, use the pragma, since you'll need to modify its list in later versions of the class.

J to L (C/C++)

These are C/C++ compiler error messages that begin with *J*, *K*, or *L*.

Error 10072 **label *Lgt* redefined**

The compiler generates this error when an attempt is made to redefine a label, in this case *Lgt*, that has already been defined.

Fix Remove or rename one of the labels.

Error 10248 **local classes shall not have member templates**

The compiler generates this error when a local class erroneously has a member template.

```
void f()
{
   struct X {
     template <class X> void f(X) {};  // error
   }
}
```

Error 10111 **local data >32k**

This error is issued when the local data totals exceed 32K. The local data, usually a declared array, is stored on the stack and has a limit of 32K.

Fix You can overcome this restriction by defining an array as static or using dynamic allocation to move the storage from the stack to the heap.

Error 10163 **local data > 224 bytes**

(Mac OS PPC) This error is caused in assembly functions which have no stack frame. In such a function there is a limit of 224 bytes of local variables.

Fix To resolve this create a stack frame, using the `fralloc`/`frfree` directives.

local variable *name* was not assigned to a register

(Mac OS PPC) The compiler generates this error when a `register` variable was named but not assigned as a `register`. In assembly functions, any variable declared `register` is guaranteed to be in a `register` and its name may be used anywhere a `register` is valid.

Fix This usually is because there are already too many `register` variables being used. Remove some previously assigned `register` variables to resolve this error.

M to O (C/C++)

These are C/C++ compiler error messages that begin with *M*, *N*, or *O*.

Error 10009 **macro *Macro* redefined**

The compiler generates this error when an attempt is made to redefine a macro, in this case `Macro, t`hat has already been defined.

Fix Remove or rename one of the macros.

Error 10012 **macro(s) too complex**

This error is signaled when a macro cannot be expanded because it is too complex (or possibly recursive).

Fix You can resolve this error by studying the macro and redesigning it with less complexity.

Error 10235 **'main' not defined as external 'int main()' function**

Undocumented at this time.

Error 10200 **method *mthd* not defined**

This error is generated you forget to define a method that was declared in the @interface.

Error 10194 **method *mthd* redeclared**

This error is generated because you attempted to declare the method `mthd` which had previously been declared.

Error 10201 **method *mthd* redefined**

This error is generated because you attempted to define a method `mthd` that was previously defined.

Error 10237 **name has not been declared in namespace/class**

This error occurs when you define something that has not been declared in it's namespace. For example,

```
namespace N {}
int N::a; // <<<error
```

Error 10173 **no parameters allowed in SOM class constructors**

The constructor for a SOM class cannot contain constructors. For more information on SOM objects, see the Metrowerks manual*C Compilers Guide* or the *SOMObjects Developer Toolkit* (IBM).

Error 10172 **no static members allowed in SOM classes**

A SOM class cannot contain static data members. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide* , or the *SOMObjects Developer Toolkit* (IBM).

Error 10129 **non-const ‘&’ reference initialized to temporary**

The compiler generates this message when the initial value for a reference type is not an lvalue of that type. The compiler will create a temporary for the initialization. However, there is no storage for this temporary, as in Listing 2.90.

See Also “not an lvalue” on page 67

Listing 2.90 **Non-const ‘&’ reference initialized to temporary**

```
long  &r = 40000;
// the proper method to use  is
long x;
```

```
long &y = x;
y = 40000;
```

Error 10183 **new SOM callstyle method *method* must have explicit 'Environment *' parameter**

If you create a SOM class that uses new IDL callstyle, each of the class's methods must contain an Environment pointer as its first argument. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide* , or the *SOMObjects Developer Toolkit* (IBM).

Fix There are two solutions:

- Add an Environment pointer to the method's argument list as its first argument.
- Use the `SOMCallStyle` pragma to declare that all of the class's methods use the older OIDL callstyle. The `SOMCallStyle` method looks like this:
  `#pragma SOMCallStyle OIDL`

Error 10050 **not a struct/union/class**

The compiler expected to find a struct, union, or class, but found a simple type instead, as in Listing 2.91.

Listing 2.91 **Not a struct**

```
long  var;
var.myfoo = 10;// error: var is not a struct
```

Error 10043 **not an lvalue**

The compiler expected an expression referring to an item, such as a variable, to which it can assign a value. Another expression was found instead.

Error 10055 **number is out of range**

The compiler signals this error when a numeric value is encountered that is out of range for its data type.

See Also For a complete list of data types supported by Metrowerks C/C++, see the Metrowerks *C, C++, and Assembler Language Reference*.

Error 10234 **object *object* redefined**

The compiler generates an error when an object is incorrectly redefined, as shown in Listing 2.92.

Listing 2.92 **Object *object* redefined**

```
void foo() {}
void foo() {}//<<< ERROR
```

Error 10198 **Objective-C type *Type* is undefined (should be defined in objc.h)**

This error is generated because you attempted to use an Objective-C type `Type` that had not been defined in the objc.h header file.

Error 10199 **Objective-C type *Type* has unexpected type**

This error is generated because you attempted to use a type `Type` that included an unexpected type for its object type.

P to R (C/C++)

These are C/C++ compiler error messages that begin with *P*, *Q*, or *R*.

Error 10127 **pascal function cannot be overloaded**

CodeWarrior does not let you overload pascal functions. The Listing 2.93, when compiled, will issue this error.

Listing 2.93 Illegal pascal function overloading

```
int f(int);
pascal void f();    //  error
```

Error 10049 **pointer/array required**

This error is issued when the compiler finds a left bracket, `[`, following a variable which is neither a pointer nor an array. The left bracket can only follow a pointer or array name.

Error 10107 **possible unwanted ';'**

(Warning Message) A semicolon was found immediately following a `while`, `if`, or `for` statement. This may cause an unintended logical error, as in Listing 2.94. This warning is signaled when the **Possible Errors** option is selected in the C/C++ Warnings settings panel.

Fix Either remove the unwanted semicolon or deselect the **Possible Errors** option in the C/C++ Warnings settings panel.

See Also For more on the **Possible Errors** option, consult the *C, C++, and Assembler Language Reference*, Chapter 2, "C and C++ Language Notes.".

Listing 2.94 Possible unwanted ';'

```
while (x < 10);// possible unwanted ';'
   printf("%d  ", x);
```

Error 10191 **'pointer to member' is not supported for SOM classes**

You cannot use take the address of a member of a class that's descended from SOMObject. For example, `&foo::bar` is not allowed if `foo` is descended from SOMObject. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide* , or the *SOMObjects Developer Toolkit* (IBM).

Error 10108 **possible unwanted assignment**

(Warning Message) This warning occurs when an assignment (= operator) occurs within a logical expression in a `while`, `if`, or `for` statement. This may be meant as an equality operation, as in Listing 2.95. This is signaled as an error when the **Possible Errors** checkbox is selected in the C/C++ Warnings settings panel.

Fix Either correct the unwanted assignment or deselect the `Possible Errors` checkbox under in the C/C++ Warnings settings panel.

See Also For more on the **Possible Errors** checkbox, consult *C, C++, and Assembler Language Reference*, Chapter 2, "C and C++ Language Notes.".

Listing 2.95 Possible unwanted assignment

```
if (x = 20) printf("OK");
    // possible unwanted assignment.
```

Error 10109 **possible unwanted compare**

(Warning Message) This warning occurs when the compiler believes it finds an unwanted comparison as in Listing 2.96.

NOTE: The error is signaled when the **Possible Errors** checkbox is selected in the C/C++ Warnings settings panel.

Fix Either correct the unwanted comparison or deselect the **Possible Errors** checkbox under the C/C++ Warnings settings panel.

See Also For more on the `Possible Errors` checkbox, consult *C, C++, and Assembler Language Reference*, Chapter 2, "C and C++ Language Notes.".

Listing 2.96 Possible unwanted compare

```
x == 1;// possible unwanted comparison
```

Error 10019 **preceding #if is missing**

This error is issued when an `#endif` directive is found without a matching `#if` directive.

Fix Examine the logic behind previous nested `#if` structures to make sure you haven't included an additional `#endif` directive.

Error 10138 **preceding '#pragma push' is missing**

The compiler generates this error when it encounters a `#pragma pop` encountered that does not have a matching, preceding `#pragma push`.

For more on the pragmas available in CodeWarrior, consult the *C, C++, and Assembler Language Reference*.

Error 10219 **preprocessor #error directive**

The compiler generates an error when it encounters the directive `#error`.

Fix

NOTE: Before fixing this error, you should check to see why this error directive was added. The directive `#error` is usually used to prevent the programmer from compiling a section of code in certain situations.

To fix the error, remove the `#error` directive.

Error 10238 **preprocessor #warning directive**

The compiler generates a warning when it encounters the directive `#warning`.

NOTE: Before fixing this error, you should check to see why this warning directive was added. The directive `#warning` is usually

used to tell the programmer that unexpected things could happen in a section of code.

NOTE: This directive is not available when the option ANSI Strict is enabled in the **C/C++ Language** settings panel.

Fix To fix the error, remove the `#warning` directive.

Error 10018 **preprocessor syntax error**

The compiler encounters an illegal preprocessor directive, as in Listing 2.97.

Fix Check the syntax of the directive in question. If the error is not apparent, it is likely being caused by a previous error.

Listing 2.97 Preprocessor Syntax Error

```
#include file
```

Error 10209 **protocol *prtcl* redefined**

This error is generated because the protocol `prtcl` was previously defined.

Error 10211 **protocol *prtcl* is already in protocol list**

The compiler generated this error because the protocol `prtcl` was previously listed in the protocol list.

Error 10210 **protocol *prtcl* is undefined**

This error is generated because the protocol `prtcl` was not defined.

Error 10225 **protocol list does not match @interface**

The compiler generates an error when the protocol list declared in the interface does not match what is defined in the implementation.

Listing 2.98 Protocol list does not match @interface

```
@protocol a
@end
@protocol b
@end
@interface A <a>
@end
@implementation A <b>//<<< ERROR: inconsistent with declaration
@end
```

Error 10205 **receiver cannot handle this message**

This message is generated because the receiver could not properly handle the message.

Error 10061 **reference to label *lbl* is out of range**

This error is given when an assembly function contains a branch whose destination is out of range, as in Listing 2.99.

Listing 2.99 Reference to label 'lbl' is out of range

```
bra.s label// error: label too far away
```

Error 10085 **return value expected**

This error is generated when a function declared to return a value, does not contain a return value. For example, the `var()` function in Listing 2.100 should return an `int` or be declared as `void`.

Fix Declare the function in question as `void`, or return a value.

WARNING! In C++ a function declared without a return value is implied to return an `int` type as in Listing 2.101.

Listing 2.100 Return value expected

```
int var() {}//error: no return value
```

Listing 2.101 Return value from main() function expected.

```
main()
{
  cout << “working”;
      //<--  no return error here.
}
```

Error 10158 **RTTI option is disabled**

This compiler error is displayed when run-time type identification is attempted when the `RTTI` pragma or the **Enable RTTI** option in the C/C++ Language settings panel is off

Fix Turn on the **Enable RTTI** option in the C/C++ Language settings panel.

Error 10187 **sizeof() is not supported for SOM classes**

You cannot use a class or object descended from SOMObject in a `sizeof()` expression. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide* , or the *SOMObjects Developer Toolkit* (IBM).

S to T (C/C++)

These are C/C++ compiler error messages that begin with *S* or *T*.

Error 10180 **SOM class access qualification only allowed to direct parent or own class**

When you invoke a method with explicit scope (such as `obj->B::func()`), the specified class (`B`) must be the same class as the object (`obj`) or a direct parent of the object’s class.

For example, if class `A` is the parent of class `B` which is the parent of class `C`, then

```
C* obj = new C;

obj->C::func();// OK: C is obj's class
obj->B::func();// OK: B is a direct parent
                //  of obj's class
obj->A::func();// ERROR: A is NOT a direct
                //     parent of obj's class
```

For more information on SOM objects, see the Metrowerks manual *C Compilers Guide* , or the *SOMObjects Developer Toolkit* (IBM).

Error 10190 **SOM class arrays are not supported**

You cannot create arrays of SOM objects. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide* , or the *SOMObjects Developer Toolkit* (IBM).

Fix Store the SOM objects some other way (such as a linked list).

Error 10170 **SOM class data members must be private**

All the data members of a SOM class must have private access. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide* , or the *SOMObjects Developer Toolkit* (IBM).

Fix Make sure you don't declare any data members in a protected or public access area of your class.

Error 10188 **SOM classes cannot be class members**

You cannot declare a SOM class as a member of another class. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide* , or the *SOMObjects Developer Toolkit* (IBM).

Fix Declare a pointers to the SOM object as a member, instead

Error 10168 **SOM classes can only inherit from other SOM based classes**

A SOM class can inherit only from classes that are descendants of SOMObject. If you use multiple inheritance, you cannot mix SOM

classes and regular classes together. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide* , or the *SOMObjects Developer Toolkit* (IBM).

Fix Make sure all the SOM class's base classes are descended from SOMObject. If you don't want to create a SOM class, make sure *none* of the base classes are descended from SOMObject.

Error 10169 **SOM classes inheritance must be virtual**

When you declare a SOM class, all its base classes must be virtual. For more information on SOM objects,see the Metrowerks manual *C Compilers Guide* , or the*SOMObjects Developer Toolkit* , published by IBM.

Fix Make sure the `virtual` keyword appears before each of the class's bases in the class's declaration.

Error 10192 **SOM class has no release order list**

(Warning Message) You created a SOM class without a `SOMReleaseOrder` list, and the **Extended Error Checking** option is on. The compiler assumes the release order is the same as the order in which the functions appear in the class declaration. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide*, or the *SOM Objects Developer Toolkit*, published by IBM.

Fix There are two ways to avoid this warning:

- In the C/C++ Warning settings panel, turn off the **Extended Error Checking** option.
- Include a `SOMReleaseOrder` pragma for the class which gives the release order for all the class's member functions.

Error 10181 **SOM class must have one non-inline member function**

A SOM class must have at least one member function that isn't inline. MacSOM uses this class to determine which translation unit implements the class. For more information on SOM objects,see the Metrowerks manual *C Compilers Guide* or the *SOMObjects Developer Toolkit* (IBM).

Fix Make sure the class contains at least one member function that isn't inline. If necessary, create an empty one.

Error 10175 **SOM runtime function *func* not defined (should be defined in somobj.hh)**

The compiler expects to find certain runtime functions in the `somobj.hh` header file, but the compiler can't find one of them. The `somobj.hh` file may have been corrupted, or you may have edited the file incorrectly. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide*, or the *SOM Objects Developer Toolkit* (IBM).

Fix Replace the `somobj.hh` header file on your hard disk with a copy from the CodeWarrior CD. Modify `somobj.hh` only if you're familiar with MacSOM.

Error 10176 **SOM runtime function *func* has unexpected type**

The compiler expects runtime functions in the `somobj.hh` header file to be defined in a certain way, but the return type for one of the functions is wrong. The `somobj.hh` file may have been corrupted, or you may have edited the file incorrectly. For more information on SOM objects, see the Metrowerks manual *C Compilers Guide,* or the *SOMObjects Developer Toolkit* (IBM).

Fix Replace the `somobj.hh` header file on your hard disk with a copy from the CodeWarrior CD. Modify `somobj.hh` only if you're familiar with MacSOM.

Error 10182 **SOM type *variable* undefined**

This error occurs when a SOM specific data type (eg Environment) cannot be found. This error usually appears because you haven't included the right header files.

Error 10226 **super class does not match @interface**

The compiler generates an error when the super class declared in the interface does not match what is defined in the implementation.

Listing 2.102 super class does not match @interface

```
@interface a
@end
@interface b
@end
@interface c : a
@end
@implementation c : b//<<< ERROR: inconsistent with declaration
@end
```

Error 10007 **string too long**

The character string in question is too long.

Fix Normally this error message is displayed because a terminating quotes mark was omitted from the string. A solution is to turn on the **Color Syntax** option in the Editor settings panel.

Error 10034 **struct/union/class member *stType* redefined**

This error appears when an attempt is made to redefine a struct, union, enum, or a class member that has already been defined.

Fix Typically this happens when you use a name you have already assigned. Remove or rename one of the struct, union, enum, or class members.

Error 10038 **struct/union/class size exceeds 32k**

This error appears when the size of a class, union, or struct is greater than 32k. A struct, class, or union, usually a declared array, is stored on the stack and has a limit of 32K.

Fix You can overcome this restriction by defining an array as static or using dynamic allocation to move the storage from the stack to the heap.

Error 10033 **struct/union/enum/class tag *stType* redefined**

This error appears when an attempt is made to redefine a struct, union, enum, or a class tag that has already been defined.

Fix Typically this happens when you use a name you have already assigned. Remove or rename one of the struct, union, enum, or class tags.

Error 10135 **template redefined**

This error is given when an attempt is made to redefine a template that has already been defined, as in Listing 2.103.

Fix Remove or rename one of the struct, union, enum, or class tags.

Listing 2.103 Template redefined

```
template <class T> class aClass { ... };

template <class T> class aClass { ... };  //  error
```

Error 10136 **template parameter mismatch**

The compiler gives this error when the member function template parameter list does not match the class parameter list, as in Listing 2.104.

Listing 2.104 template parameter mismatch

```
template <class T> class aClass { void f() };
template <class T,class U>
  void aClass<T>::f() { ... }; //  error
```

Error 10215 **template too complex or recursive**

The compiler gives this error when there are too many recursive template expansions.

Fix Decrease the complexity of your algorithm.

Error 10052 **the file *filename* cannot be opened**

The compiler cannot find a file name provided in an `#include` directive, as in the second `#include` directive in Listing 2.105.

Fix It is possible that the file name specified in the `#include` directive is spelled wrong or is not on a valid access path. Switch to the Finder and use the **Find** command to find the file in question.

It is also possible that the `#include` file is specified as a system include, `<..>`, when it should be specified as a user include, `"..."`. If this is the case, select the **Always Search User Paths** checkbox in the Access Path settings panel.

See Also For more on access paths and the option **Always Search User Paths**, consult the *C Compilers Reference*

Listing 2.105 File cannot be opened

```
#include "AbstractHeader.h"
#include "Foo.h"// This file doesn't exist
#include <QDoffscreen.h>
```

Error 10167 **the parameter(s) of the *functionname* function must be immediate value(s)**

Undocumented at this time.

Error 10048 **too many initializers**

This error is given when the number of initialization values is greater than the number of items specified in the declaration of the initialized structure.

Fix Typically you encounter this error when initializing elements in an array, structure or class and you attempt to assign more values than elements declared. Adjust the number of elements in the array, class or structure or use the correct number of initializers.

Error 10011 **too many macro arguments**

The compiler issues this error when an attempt is made to define a macro with more than 32 arguments, as in Listing 2.106.

Listing 2.106 **Too many macro arguments**

```
#define macro(arg1,...,arg33) ...
```

Error 10146 **type mismatch *A_type* and *B_type***

The compiler issues this error when expects to find one data type, but finds another instead.

This error may also occur if one of your functions has the same name as a Metrowerks macro. For example,Listing 2.107 gives a type mismatch error.

Fix If the data types are involved are castable, typecast the offending data type to the correct type.

In Listing 2.107, `Length` is a macro defined in the precompiled headers `MacHeaders68K` and `MacHeadersPPC`:

```
#define Length(s) (*(unsigned char *)(s))
```

To fix this error, rename your variable to something other than a pre-defined macro.

Listing 2.107 **Type mismatch**

```
class MyLine {
public:
        MLine(double theLen);
        double Length(void);
```

Error 10233 **typename redefined**

The compiler generates an error when a typename is incorrectly redefined, as shown in Listing 2.108.

Listing 2.108 Typename redefined

```
struct B;
typedef int B;//<<<
```

U to Z (C/C++)

These are C/C++ compiler error messages that begin with *U*, *V*, *W*,, *X*, *Y*, or *Z*.

Error 10195 **undefined method *mthd***

This error is given when a method name `mthd` was used but had not been defined.

Error 10041 **undefined identifier *var***

This error occurs when an identifier is used that has not been defined.

Fix This is often caused from a variable not being declared within the scope it appears in. Also, check for spelling errors.

Error 10060 **undefined label *Lbl***

The compiler generates this error when a `goto` statement specifies a label that has not been defined within the function.

Fix Remove the `goto`, or create the necessary label within the scope where the error occurs.

Error 10005 **undefined preprocessor directive**

This error is signaled when a preprocessor directive not recognized by Metrowerks C/C++ is used.

Fix A list of preprocessor directives recognized by Metrowerks C are listed in the *C, C++, and Assembler Language Reference*. Consult this list to make sure the directive you are using exists and is spelled correctly.

Error 10003 **unexpected end of file**

The end of a source code file was reached before a language item was completed.

Fix This error may be caused by a misplaced or unbalanced right brace. Check the penultimate line of the source code file in question. If the error is not apparent, fix all previous errors and recompile.

Error 10013 **unexpected end of line**

The end of a source code line was reached before a language item was completed.

Fix This error may be caused by anything from a misplaced semicolon to a missing symbol. Check the source code line in question. If the error is not apparent, fix all previous errors and recompile.

Error 10021 **unexpected token**

This error occurs when the compiler finds an unexpected token.

Fix If the error is not apparent, it is likely being caused by a previous syntax error or missing symbol. Fix all previous errors and recompile.

Error 10091 **unimplemented C++ feature**

The compiler generates this error when it encounters a C++ feature that is not yet supported by Metrowerks C++.

Error 10162 **unknown assembler instruction mnemonic**

This error message is used to report an illegal instruction name.

Fix Correct the mnemonic in the assembler instruct

Error 10207 **unknown message selector**

This error is generated because the message selector was not declared or defined in any of the object hierarchy.

Error 10020 **unterminated #if / macro**

This error is displayed when an `#if` directive is found with no matching `#endif` directive, or a macro definition is not complete.

Error 10004 **unterminated comment**

The end of a source code file was reached before a comment was completed.

Error 10068 **value is not stored in register**

Undocumented at this time.

Error 10086 **variable *var* is not initialized before being used**

The compiler encounters an expression using a variable that has neither been assigned a value nor initialized. For example the code in Listing 2.109 would cause this error.

Fix Initialize or assign a value to the variable before using it in an expression.

Listing 2.109 **Variable is not initialized before being used.**

```
static int f()
{
  int i;
  return i;
}
```

Error 10083 **variable *var* is not used in function**

(Warning Message) A variable declared in a function is not used in the function body. This warning is signaled as an error if the **Unused Variables** checkbox is selected in the C/C++ Warnings settings panel.

Fix Either remove the unused variable, or deselect the **Unused Variables** checkbox.

See Also For more on the **Unused Variables** checkbox, see the *C Compilers Reference*

Error 10120 **virtual functions cannot be pascal functions**

This error appears when a virtual function is declared as a `pascal` function. For example, in Listing 2.110, the class `var` illegally declares `func()` as a `pascal` function.

Listing 2.110 Virtual functions cannot be pascal functions

```
class aClass {
  pascal int func(); // illegal declaration
};
```

3

Pascal Compiler Error Messages

This chapter gives an alphabetical list of the most common compiler errors which may be encountered while using Metrowerks CodeWarrior compilers for both the PowerPC-based and 68K-based Macintosh when using the Pascal programming language.

Pascal Compiler Errors

In this list, errors with variable initial text (such as a class or function name) come first. Errors beginning with a non-alphabetic symbol character come next. After that, errors are listed alphabetically.

Symbol Names (Pascal)

These are Pascal compiler error messages that begin with a symbol name, the name of a variable or function.

Error 12182 ***field*** **not visible in this scope**

This error is given because the field is not visible in the current scope. This mostly occurs with protected and private object fields.

Fix Make sure the field you are trying to access is not private or protected.

Listing 3.1 ***field*** **is not visible in this scope**

```
program test;

type
  obj = object
```

```
      protected
        a: integer;
    end;

var
  o: obj;

begin
  new(o);
  o.a :=1;
end.
```

Error 12052 ***identifier* must be a type**

This is an error because this identifier *identifier* was not declared as a type.

***identifier* must be a type but was defined as a [*constant identifier* | *variable identifier* | *procedure name* | *function name*]**

The identifer used as a type in the parameter declaration or in the function return is not declared as a type, but as a constant, a variable, a procedure, or a function.

Fix Change the parameter or function return type identifier.

Error 12133 ***param* could not be assigned to a register**

This PowerPC assembler error is given because the compiler cannot assigned this variable or parameter to a register.

Error 12157 ***var* doesn't start a variant list**

You are initializing a variant record with a value that is not in the bounds defined for this variant record, as shown in Listing 3.2.

Fix Make sure the value used for choosing the variant part is in the bounds of one of the variant record case labels.

Listing 3.2 Initializing a variant record

```
rect = RECORD
  foo : integer;
  CASE bar : boolean OF
    true : (a,b: char);
    false: (i:integer);
END;

rec1: rect = (foo:1, bar:true, a:'a', b:'b');
    { OK }
rec2: rect = (foo:2, a:'a', b:'b');
    { ERROR: Didn't initialize bar }
rec3: rect = (foo:3, bar:false, a:'a', b:'b');
    { ERROR: If bar is false, you must}
    {       initialize i, not a and b. }
```

Punctuation (Pascal)

These are Pascal compiler error messages that begin with punctuation marks.

Error 12070 '.' expected

A period (`.`) is the last delimiter of the source code of the main or a unit file. It is also the delimiter between a record variable name and its associated fields, between a unit name and its routine call, and between a method name and its object type in the method declaration.

Fix Possible fixes include:

- Make sure the final `END` statement is properly spelled.
- Make sure the final `END` statement is followed by a dot (`.`) instead of a semi-colon (`;`).

Listing 3.3 '.' expected

```
program test;

type
  t_obj = object
    i:integer;
    procedure a;
  end;

  procedure t_obj{.}a;
  begin
  end;

begin
end{.}
```

Error 12061 '..' expected

This error occured beause the compiler can't translate an improper array declaration. Ellipsis are expected between low and high values, and between range type or range enumeration lists. Pascal array declarations need beginning and end subscripts separated by two dots (`..`) (Listing 3.4).

Listing 3.4 Example array declarations

```
TYPE
  GoodArrayType = ARRAY [1 .. 10] OF REAL; { OK }
  BadArrayType = ARRAY [10] OF REAL; { Error }
```

Error 12059 ',' expected

The compiler can't find a comma (`,`) to separate parameters in a routine declaration or call.

Listing 3.5 Example comma error

```
program test;
```

```
var
  s: string;

begin
  s:=COPY(s,1 2); {there is a missing ',' between the routine}
                  {parameters}
end.
```

Error 12074 **';' expected**

A semi-colon is missing at the end of a statement (Listing 3.6).

Fix Add the missing semicolon.

> ***For beginners:*** A semicolon ends every Pascal statement (except the last statement, which ends with '.').

Listing 3.6 Example semi-colon error

```
VAR
  b : CHAR;       { OK }
  c : REAL        { Error }
  d : INTEGER;
```

Error 12058 **':' expected**

The compiler expected a colon (:) to denote the data type of an identifier (variable, object, parameter, or routine).

Fix Replace the erroneous operator with the colon operator.

Error 12057 **':=' expected**

You've attempted to assign a value to a variable without the assignment operator - the assignment operator (:=) is expected (Listing 3.7).

Fix Replace the erroneous operator with the assignment operator.

Listing 3.7 Example of an incorrect variable assignment

```
  c := 3; { OK }
  c = 3;  { Error, "=" is a comparison operator }
```

Error 12063 **'=' expected**

The compiler expected the equality comparison (=) operator in a boolean expression.

Fix Replace the erronous operator with a '='.

Error 12067 **'[' expected**

A left bracket delimiter is expected after the ARRAY keyword.

Fix Add the expected left bracket.

Error 12071 **']' expected**

A closing right bracket is missing (Listing 3.8).

Fix Add the expected right bracket.

Listing 3.8 Example of a missing right bracket

```
VAR
  c : ARRAY [1 .. 10] OF INTEGER;

BEGIN
  c[1] := 6; { OK }
  c[2  := 28; { Error }
```

Error 12068 **'(' expected**

The compiler expects to find an opening left parenthesis to begin an expression.

Fix Add the expected left parenthesis.

Listing 3.9 Example of '(' expected error

```
program test;

type
  t_rec = record
    case b:boolean of
      false: );  // variant cases delimiters are parenthesis
  end;

const
  c_0 : integer = integer 0); // type cast delimiters are
                              // parenthesis

begin
end.
```

Error 12072 ')' expected

The compiler expects to find a closing right parenthesis to end an expression (Listing 3.10).

Fix Add the missing `')'`.

Listing 3.10 Example of a missing closing parenthesis

```
  b := cos(c);  { OK }
  a := sin(b;  { Error }
```

Error 12125 '...' can't be used in this context

You are using an ellipsis incorrectly. Ellipsis cannot be used in a procedural type definition.

Listing 3.11 '...' can't be used in this context

```
program test;

type
  proc = procedure (a:integer; ...);  //ERROR
```

```
begin
end.
```

Error 12192 **{$endc} or {$endif} expected**

This error occurs because `{$end}` is not a valid directive for closing an `{$if}` or `{$ifc}` conditional compilation directive.

Fix Use `{$endif}` in the case of `{$if}` and `{$endc}` in the case of `{$ifc}`.

Listing 3.12 Example of "{$endc} or {$endif} expected" error

```
program test;

{$ifc true}
{$end}        //ERROR

begin
end.
```

Error 12164 **$error**

(Warning) Your code contains the `$error` directive, which prints a user-defined warning message to the Messages Window.

Listing 3.13 Example of $error

```
program test;

begin

{$error My error}

end.
```

A to C (Pascal)

These are Pascal compiler error messages that begin with *A*, *B*, or *C*.

Error 12184 **actual declaration for *identifier* missing**

An error occurred because an actual declaration for this anonymous type was not given in the implementation of the unit.

Error 12159 **actual parameter's size is too small, could mangle memory**

(Warning) The string you're passing to a routine is smaller than the routine expects. Since the string parameter is a VAR parameter, copying the string could mangle memory.

Listing 3.14 Actual parameter size is too small, could mangle memory

```
program test;

type
  str10 = string[10];
  str20 = string[20];


  procedure a(var x: str20);
  begin
  end;

var
  s10: str10;
  s20: str20;

begin
  a(s10);
end.
```

Error 12030 **already declared**

The underlined symbol is declared more than once in the current scope (Listing 3.15).

Fix Remove the extra declaration.

Listing 3.15 Example of the already declared error

```
program test;

type
  t_char = char;
  t_char = ^char; // 't_char' type is already declared

var
  c: t_char;
  c: char;        // variable 'c' is already declared

begin
end.
```

Error 12161 **already defined macro**

This error is generated when you try to redefine a macro. An example of this error is below in Listing 3.16.

Listing 3.16 Already defined macro

```
program test;

{$definec x 1}
{$definec x 2}

begin
end.
```

Error 12188 **array element type cannot be a schema**

An error occured because the component type of an array cannot be a type derived from a schema.

Listing 3.17 Array element type cannot be a schema

```
program test;

type
  arrElement(i:integer) = array [1..i] of integer;
  arrType = array [1..10] of arrElement(5);

begin
end.
```

Error 12110 **assignment to loop index variable**

(Warning) The compiler is warning you that the variable used as an index in a `FOR...DO` loop is assigned a value in the loop. Doing so might prevent the loop from terminating (Listing 3.18).

TIP: To make the compiler warn about this condition, select **Modified For-loop Indexes** in the Pascal Warnings preferences panel.

Fix Remove the assignment to the index variable, or use another variable.

Listing 3.18 Example of loop index assignment

```
program test;

var
  i: integer;

begin
  for i :=1 to 100 do
  begin
    i:=1;
  end;
end.
```

Error 12085 **bad precompiled unit format**

The compiler gives this error because the precompiled unit this project uses is corrupt, or because the object code from an earlier compilation is corrupt.

Fix To correct this error:

- In the case of a bad precompiled unit, recompile the precompiled unit project.
- In the case of corrupted binaries, choose **Remove Object Code** from the **Project** menu, and recompile the project.

If the problem persists, please contact Metrowerks Technical Support.

Error 12001 **bad symbol**

[Obsolete] This error has been replace by the error bad precompiled unit format.

Error 12151 **'BEGIN' expected**

The compiler expected to find the beginning of a compound statement, and it didn't find one.

Listing 3.19 **'BEGIN' expected**

```
program test;

end.
```

Error 12189 **cannot assign to a record with schema array type fields**

You cannot perform an assignment to a record type variable that contains fields which are array types derived from schemata. Listing 3.20 shows code that demonstrates this error.

Listing 3.20 Cannot assign to a record with schema array type fields

```
program test;

type

  schemaType(i:integer) = array [1..i] of integer;

  rec = record
      arr :schemaType(10);
    end;

var
  r1, r2: rec;

begin
  r1:=r2;
end.
```

Error 12187 **cannot be a schema**

Only non-packed records and arrays can be used to declare a schema. You cannot use a variable of a type derived from a schema as an argument to an open array type parameter. You also cannot use a schema as a function return type. Object type data members cannot be types derived from schemata.

Error 12028 **cannot be a var parameter**

The compiler gives this error became the underlined parameter in the routine call is not a variable and it should be.

Fix Assign routine call parameter value into a variable and used the-variable to call the routine.

Listing 3.21 Cannot be a var paramter

```
program test;

    procedure a(var c:char);
```

```
    begin
    end;

begin
   a('a'); // should be replaced with: c:='a'; a(c);
end.
```

Fix Assign routine call parameter value into a variable and used the variable to call the routine.

Error 12086 **cannot find file *filename***

This error is generated when the compiler could not find a file while compiling or linking a project.

Fix Some suggestions for helping the compiler find your file(s):

- See if you misspelled the filename.
- Make sure you have specified the correct access path in the project settings. Choose **(Target) Settings** from the **Edit** menu, where (Target) is the name of your project. Click **Access Paths** to see the panel where you specify your project's access path settings.

For beginners: Complete information on the Access Paths settings panel is available in the CodeWarrior manual *IDE User Guide*.

Error 12171 **cannot mix different styles of conditional compilation directives**

This error is generated because you tried to mix Borland and MPW style conditional compilation directives.

Fix Use the same style of conditional compilation directives.

Listing 3.22 cannot mix different styles of conditional compilation directives

```
program test;

{$ifc true}
{$endif}{Error: should be $endc, not $endif}

{ $if true}
{ $endif}

begin
end.
```

Error 12173 **cannot mix dynamic arrays and non-local gotos**

This error is generated because you tried to have an open array parameter for a nested routine that uses `exit`. This error only occurs when you are targeting a PowerPC processor.

Listing 3.23 Cannot mix dynamic arrays and non-local gotos

```
{$if POWERPC}
program test;

  procedure a(arr:array of integer);
    procedure b(arr2:array of integer);
    begin
      arr2[0]:=1;
      exit(a);
    end;
  begin
    b(arr);
  end;


var
  arr:array [0..100] of integer;
begin
  a(arr);
```

```
end.
{$endif}
```

Error 12156 **cannot mix optimized and unoptimized classes**

The compiler gives this error because you tried to mix optimized and unoptimized classes

Fix To avoid this error, make sure the option **Optimize Class Hierarchy** in the Pascal Language settings panel is either on or off for all Pascal units (project sources and Pascal unit libraries).

Error 12155 **cannot mix pointer based and handle based classes**

This error is generated because you tried to mix pointer based and handle based classes.

Fix To avoid this error, make sure 'pointer based objects' option in Pascal Language pref panel is either on or off for all units (project sources and Pascal librairies) included in your project.

Error 12174 **cannot mix routines from different IO models**

This error was generated because this routine cannot be used with this IO model.

Fix Make sure that all standard routine you uses are compatible with the IO model you have selected.

Listing 3.24 Example of cannot mix routine from different IO models error

```
program test;

{$pragma IOmodel turboIO}

  procedure p1;
  begin
    exit(p1);
  end;
```

```
begin
end.
```

Error 12041 **cannot mix short-circuit and non short-circuit logical operations**

An error was generated because it is illegal to mix short-circuit operators (`&,|`) with non short-circuit operators (`and,or`) within the same expression.

Fix Decompose the expressions into two or more expressions.

Error 12181 **cannot nest an object method definition**

An error occurred because you have attempted to nest the definition for a method inside the definition of another method (for example, a procedure or function), as in Listing 3.25.

Listing 3.25 Cannot nest an object method definition

```
type
  Widget = object
      procedure Run;
  end;

procedure Foo;
  procedure Widget.Run; //ERROR
  begin
  end;
begin
end;
```

Error 12128 **cannot override data field *ident***

You already used this identifier *ident* as a data field for this class or one of its ancestors. Data fields cannot be overridden.

Listing 3.26 Cannot override data field *ident*

```
program test;

type
  obj1 = object
      a,b:integer;
    end;

  obj2 = object (obj1)
      a:real;
    end;

begin
end.
```

Error 12042 **cannot pack this type**

An error message was given because this type declaration cannot be made a packed type.

Fix The keyword `packed` is legal only with set, array, record and file types. Remove the packed keyword.

Listing 3.27 Cannot pack this type

```
program test;

type
  t_enum = (red, blue, green);
  t_illegal = packed t_enum;    //ERROR

begin
end.
```

Error 12160 **cannot use '^' to start a procedural type**

The compiler gives this error because the character '^' cannot be used in a procedural type definition.

Listing 3.28 cannot use '^' to start a procedural type

```
program test;

type
  proc1 = ^procedure; {Error}
  proc2 = procedure;  {OK}

begin
end.
```

Error 12186 **cannot use formal discriminants in enumeration lists**

You cannot use the formal discriminants in a schema declaration as enumerated identifiers.

Error 12169 **can't be a routine type**

You cannot assign a parameter of a routine type to a variable of a procedural type. The parameter may represent a nested routine, but variables of a procedural type cannot represent nested routines.

Listing 3.29 Can’t be a routine type

```
type intfunc = function ( i : integer ) : integer;

procedure foo ( function a ( i : integer ) :
    integer );
var x : intfunc;
begin
    x := a; // ERROR
end;
```

Error 12167 **can’t be an open array base type**

You cannot declare an open array (that is, an array with no range) of this type (for example, an open open array of files.)

Listing 3.30 Can't be an open array base type

```
program test;

  procedure a (arr: array of file);
  begin
  end;

begin
end.
```

Error 12145 **can't initialize an occurrence of this type**

You cannot perform static initialization on a variable of this type.

Listing 3.31 can't initialize an occurrence of this type

```
program test;

type
  class = object
      a,b: integer;
    end;

var
  o1 : class;

const
  o2: class = o1;

begin

end.
```

Error 12117 **can't override *objectMethod***

The compiler generates this error message because this method cannot be overridden.

Fix Make sure the method you tried to override is inherited from a superclass or that it's name is not misspelled.

Listing 3.32 can't override *objectmethod*

```
program test;

type
  obj = object
      procedure a; override;
    end;

begin
end.
```

Error 12177 can't precompile this file

This error is given because you tried to precompile a program file or a unit with an `IMPLEMENTATION` part.

Fix Make sure you precompile your file or unit with no `IMPLMENTATION` part.

Error 12009 case constant defined more than once

A value in a `CASE` statement is repeated (Listing 3.33).

Fix Possible fixes include:

- Change the case constant value.
- Merge the case constant code that has the same value.

Listing 3.33 case constant defined more than once error

```
...
CASE i OF
  1     : x := Sin(x);
  2     : x := Cos(x);
  3     : x := Exp(x);
```

```
  2     : y := 0.0;
        { Error: 2 is already defined. }
  ...
```

Error 12143 **class *class name* already declared EXTERNAL**

This class is already declared as EXTERNAL. It cannot be redeclared.

Fix Remove the new class declaration, or remove the external class declaration.

Listing 3.34 Example of class *classname* already declared EXTERNAL error

```
program test;

type
  obj = object; external;

  obj = object;
      a,b: integer;
    end;

begin
end.
```

Error 12144 **class *identifier* was declared FORWARD or EXTERNAL**

An ancestor class was declared FORWARD or EXTERNAL and hasn't been fully defined yet. You must fully define a class before inheriting from it.

Listing 3.35 Class *identifier* was declared FORWARD or EXTERNAL

```
program test;

type
  obj1 = object; external;

  obj2 = object (obj1)
```

```
      a,b: integer;
    end;

begin
end.
```

Error 12080 **compiler restriction**

This language feature is not supported by the Metrowerks Pascal compiler.

Error 12139 **CONST parameter cannot be modified**

You tried to assign a value to a constant parameter.

Fix Use a temporary variable.

Listing 3.36 **CONST parameter cannot be modified**

```
program test;

  procedure a(const x: integer);
  begin
    x:=10;  // error
  end;

begin
end.
```

Error 12138 **CONST parameter cannot be passed to a VAR parameter**

You cannot pass a `CONST` parameter to a `VAR` parameter since the compiler cannot ensure that the `CONST` parameter will remain unchanged.

Fix Use a temporary variable.

Listing 3.37 CONST parameter cannot be passed to a VAR parameter

```
program test;

  procedure a(var x:integer);
  begin
    x:=0;
  end;

  procedure b(const x:integer);
  begin
    a(x);
  end;

begin
end.
```

Error 12033 **constant overflow**

An error was encountered when the value represented by this constant exceeds the internal representation available. An example would be trying to fit `4000000000` into a `longint`.

Fix Two possible fixes include:

- Look for a set base type which is too big.
- Look for a constant which exceeds the maximum value of its variable type.

Listing 3.38 Constant overflow

```
program test;

type
  t_set = set of 0..2048;  // set base type should be within
                           // 0..2047 limit range

begin
end.
```

Error 12037 **constant string too long**

The compiler generated this error message because the literal string contains to many characters. The string's length cannot be more than 255 characters.

If this doesn't seem to be the case, you may be concatenating two strings so that the combined result is a string that exceeds the 255 character limit.

Fix Decrease the length of your string(s), or separate them into chunks smaller than 256 characters.

D to F (Pascal)

These are Pascal compiler error messages that begin with *D*, *E*, or *F*.

Error 12103 **division by 0**

You have attempetd a division operation with 0 as the divisor in a constant expression in an assembler routine . Division by 0 is illegal.

Error 12060 **'DO' expected**

The `DO` keyword is missing in a `WHILE`, `WITH`, or `FOR` statement (Listing 3.39).

Fix Add the missing `DO` keyword.

Listing 3.39 **'DO' expected error**

```
FOR a := 1 TO 10 DO { OK }
  b := b + a;

WHILE b > 100 { Error: no DO keyword }
  b := b DIV 2;
```

Error 12062 **'END' expected**

An `END` keyword is expected to close a record type, a case, a procedure or function, a program or unit, a main or block statement.

Fix Possible fixes include:

- Make sure the `END` statement is properly spelled or that the `END` statement intended to finish a `BEGIN–END` statement block is not "used" by another `BEGIN–END` block.
- If the keyword is missing, add the missing `END`.

Error 12137 **end of line expected**

This assembler error is given because the compiler expected to find the end of the line but found another symbol. There is unnecessary text at the end of this line.

Error 12154 **enumerations have a maximum of 32,767 elements**

This error is given because this enumerated type excceds the maximum allowed element, which is 32,767.

Listing 3.40 Enumerations have a maximum of 32,767 elements

```
program test;

type
  enum = (element1, element2, element3, .... element32766,
element32767, element32768);

begin
end.
```

Error 12163 **error in macro definition**

This error is generated when you try to use an improperly defined macro.

Fix Make sure you don't have recursion in your macro.

Listing 3.41 Error in macro definition

```
program test;

{$definec a(x)  a}

begin
end.
```

Error 12132 **exit statement needs a routine name**

The `Exit` procedure requires an argument: the name of the routine to exit or the `PROGRAM` to exit the program.

Error 12056 **expression type must be boolean**

A non-boolean expression is used where a boolean expression is expected as in Listing 3.42.

Fix Any conditional expression (such as the expressions `IF`, `WHILE` and `REPEAT`) should be of boolean type. Be sure the condition type is boolean, or type cast to boolean if necessary.

Listing 3.42 Expression type must be boolean error

```
program test;

const
  c = 0;

var
  l: longint;

begin
  if c then   // should be: if boolean(c) then
    ;
  if l then   // should be: if boolean(l) then
    ;
  while c do  // should be: while boolean(c) do
```

```
    ;
end.
```

Error 12029 **extra ‘,’ in parameters declaration**

An error was given when the routine call contains an extra comma (,) in the argument list as in Listing 3.43.

Fix Remove the extra comma.

Listing 3.43 Example of extra , in parameters declaration

```
foo(bar,);
```

Error 12064 **‘FILE’ expected**

An error was given because this parameter must of a file type. A file pointer variable is expected in I/O routine calls.

Fix Be sure I/O routines are called with a file pointer variable.

Error 12021 **file is not allowed in this context**

This is an error because this component defines a file which is illegal in this context, as in Listing 3.44.

Fix Look for illegal use of a file type in an assignment, a parameter list or a definition.

Listing 3.44 File not allowed in this context

```
program test;

type
  t_file = file of file;// illegal component for a structured file

  function a : Text;
  var
    f: Text;
```

```
  begin
    a:=f;              // illegal assignment
  end;

  procedure b(p1:file);     // cannot have file type parameter
  begin
  end;

begin
end.
```

Error 12191 **formal discriminants cannot be used as parameters to function calls**

You cannot use the formal discriminants in a schema declaration as actual parameters to a procedure or function call.

Error 12176 **found a cycle involving *param***

This error is given because the compiler found a cycle in the "uses" list (that is, two or more units use each other).

For example,

```
//  unit A uses B, unit B uses A;              cycle A->B->A.
//  unit A uses B, unit B uses C, unit C uses A;cycle A->B->C->A.
```

Listing 3.45 Found a cycle involving *param* example

```
unit foundcycle;

interface

uses foundcycleb;

implementation

end.
```

Error 12095 **function already has a stack frame**

This assembler error message is given when a stack frame is already defined for a routine.

Fix Don't try to create more than one stack frame in a routine.

Listing 3.46 Function already has a stack frame

```
program test;

  procedure a;asm;
  begin
    fralloc;
    fralloc;   {ERROR: extra stack frame definition}
  end;

begin
end.
```

Error 12090 **function doesn't return any value**

(Warning) This error occurs when a function does not have an assignment statement that assigns a value to the function name identifier (Listing 3.47). To specify the value a function returns, the function's identifier must be assigned a value as if it were an ordinary variable.

Fix Possible fixes include:

- Add an assignment statement to the function identifier.
- Change the function definition to a procedure definition.

Listing 3.47 Function doesn't return any value errors

```
FUNCTION InchesToCms(inches : REAL) : REAL;
VAR
  temp : REAL;

BEGIN
```

```
 { Error: no assignment to InchesToCms identifier }
  temp := inches * 2.54;
END;

FUNCTION ChickenCount(hatchedEggs : INTEGER) :
    INTEGER;
BEGIN
  { Error: no assignment if condition is false }
  IF hatchedEggs = 0 THEN
    ChickenCount := 0;
END;
```

Error 12096 **function has no initialized stack frame**

This assembler error is given because this operation required an initialized stack frame for the routine.

Fix Possible fixes include:

- You must define a stack frame if you wish to use local variable in the routine.
- Don't try to release a stack frame when one hasn't been initialized.

Error 12049 **function incorrectly used as a procedure**

This error occurs when a function is incorrectly used as a procedure (that is, the function call is not assigned to a variable).

Fix Possible fixes include:

- Replace the procedure call by a function call.
- Replace the function declaration or definition by a procedure declaration or definition.
- Set the pragma `noreturn` to on to allow function calls to be used like routine calls.

Listing 3.48 Example of function incorrectly used as a procedure error

```
program test;

  function b:char;
  begin
    b:=#0;
  end;

begin
  b();
end.
```

G to I (Pascal)

These are Pascal compiler error messages that begin with *G*, *H*, or *I*.

Error 12115 goto a label enclosed in a 'FOR' statement.

This is an error because this label is the target for a `goto` statement from outside the `FOR` loop as in Listing 3.49.

Listing 3.49 Example of goto a label enclosed in FOR statement.

```
goto 1;
for <var> := <expr> to <expr2> do
1:
```

Error 12114 goto a label enclosed in a 'WITH' statement

This error message is given when the label is the target for a `goto` statement from outside the `WITH` statement. An example of this error is in Listing 3.50.

Listing 3.50 Example of goto a label enclosed ina WITH statement.

```
goto 1;
with <expr> do begin
  ...
```

```
  1:
  ...
  end;
```

Error 12113 **goto between 'CASE' legs.**

(Warning) This message is given when the goto's target is inside another CASE leg, as in Listing 3.51. The **Branching Between Case legs** option in the Pascal Warnings panel must be turned on in order to receive this warning message.

Listing 3.51 Example of goto between CASE legs.

```
program test;

label 1;

var
  i,j: integer;

begin
  j:=0;
  goto 1;
  for i:=1 to 10 do
  begin
1:  j:=j+i;
  end;
  goto 1;
end.
```

Error 12112 **goto between 'IF' and 'ELSE' parts.**

(Warning) This warning is generated when the goto's target is between an IF and ELSE, as in Listing 3.52.

NOTE: The "Branching Between If & Else" option of the Pascal Warnings panel must be turned on in order to receive this warning message.

Listing 3.52 Example of goto between IF and ELSE parts.

```
program test;

label 1;

begin
  if true then
    goto 1
  else
1:  writeln ('error');
end.
```

Error 12065 **identifier expected**

The error message is given when the compiler expected an identifier instead of the erronous or missing token.

Fix Add the missing identifier.

Listing 3.53 Identifier expected

```
program test;

  procedure a( :char); // missing parameter name
  begin
  end;

begin
end.
```

Error 12099 **illegal addressing mode**

This assembler message is generated because this addressing mode is illegal for the operation.

Listing 3.54 Illegal addressing mode

```
program test;
```

```
procedure a; asm;
begin
  move d0,somewhere; {ERROR:”somewhere” is an invalid address :-)}
end;

begin
end.
```

Error 12127 **illegal cast, size mismatch**

You tried to cast a variable of a structured type to another structured type of a different size. The structured types must be of compatible size.

Listing 3.55 **illegal cast, size mismatch**

```
program test;

  procedure a(var x:longint);
  begin
    x := 111;
  end;

var
  b:char;

begin
  a(longint(b));
end.
```

Error 12014 **illegal casting**

This is an error because of an inappropriate attempt to use a value of one data type as another data type (Listing 3.56).

Fix Remove the type casting, and call a routine that converts the value into the desired type.

Listing 3.56 Illegal casting error

```
program test;

  procedure a;
  var
    i: integer;
    r: real;
  begin
    i:=integer(r); // should write a routine that converts real
  end;             // values into integer ones instead

begin
end.
```

Error 12020 illegal declaration

This is an error because it is illegal to use this declaration in this context as in Listing 3.57.

Listing 3.57 Example of Illegal declaration.

```
unit illegaldeclaration;

interface

  procedure a( ... );// should at least has 1 parameter before
                     // the ellipsis
  procedure b; asm;  // cannot declare assembler routine in the
                     // interface part

implementation

  procedure b; asm;// (...but it is ok in the implementation part)
  begin
  end;

  procedure w;
    procedure x; c; external;// cannot declare a nested 120
                             // routine
```

```
  begin
  end;

end.
```

Error 12036 **illegal format for real constant**

An error was given because this number doesn't correspond to the notation accepted for a floating point number.

Fix Change the real number constant's notation so that its syntax is correct.

Listing 3.58

```
program test;

var
  r: real;
begin
  r:=1.d3;   {ERROR: real numbers cannot contain alphabetic}
             {       characters                             }
end.
```

Error 12023 **illegal function result type**

This error is given when a function cannot return an item of this type, as in Listing 3.59.

Listing 3.59 **Example of illegal function result type.**

```
function foo : FILE;
```

Error 12094 **illegal instruction for this processor**

This error is given because the compiler doesn't recognize this assembler instruction.

Fix Make sure this assembler instruction is available or supported for the processor you are targeting.

Error 12015 **illegal label**

The label name is not valid. Valid label name are identifier that follow Pascal naming conventions and is any integer between 0 and 9999.

Listing 3.60 Example of illegal label error

```
program test;

  procedure a;
  label
    -3,10000;
  begin
  end;

begin
end.
```

Error 12136 **illegal operand**

This assembler error is given beause the compiler doesn't recognize this operand.

Fix Make sure this operand is a previously defined symbol.

illegal operand for this processor

This assembler error message is given because this operand is not accepted for this instruction.

Error 12038 **illegal operand type**

This is an error because this operator doesn't allow operands of this type. The two operands don't have the same type, or one or both of them is an illegal type for this operator.

Fix Possible fixes include:

- Type cast the illegal type.
- Change the operand type.

Listing 3.61 Illegal operand type

```
program test;

var
  c: char;
  r: real;
begin
  c := c and r;  //ERROR: operands don't have the same type
end.
```

Error 12102 illegal operands for this processor

This assembler error message is generated because this kind of operand is not accepted by the processor.

Error 12016 illegal operation

The statement contains an illegal mathematical operation, as a zero division for example.An error is generated because this operation cannot be performed, as in Listing 3.62.

Listing 3.62 Example of illegal operation.

```
i/0 (division by literal zero)
```

Error 12170 illegal operation on a file

An error is generated because this operation is not allowed for a file or files of this type.

Error 12098 illegal register list

This inline assembler error is given because this assembler instruction doesn't accept this register list.

Fix Possible fixes include:

- Don't mix data register (`d0` to `d7`) with address register (`a0` to `a7`) in a register list.
- Make sure the register list is valid. For example, `d0-d7` is valid but `d5-d9` is not valid.

Listing 3.63 Illegal register list

```
program test;

var
  a,b:integer;

  procedure x; asm;
  begin
    movem d0-a2, (a7) {ERROR: mixed data registers}
                      {       and address registers}
  end;

begin
end.
```

Error 12172 illegal routine specification

This error was generated because this routine delcaration is invalid. The uses of `STDCALL`, `DLLIMPORT` and `DLLEXPORT` in a routine declaration for target other than Win32 cause this error.

illegal set element type

An error is given when the type cannot be used as a base to construct a set, as in Listing 3.64.

Listing 3.64 Example of illegal set element type.

```
set of real
```

Error 12017 **illegal set element value**

This error is given because this element is not of the same type as the set base type.

Fix The set's element can only be of scalar type.

Listing 3.65 Example of illegal set element value error

```
program test;

type

  aSet = set of 1..100;

var
  s : aSet;
begin
  s := [1.1];
end.
```

Error 12018 **illegal statement**

This is an error because the statement is illegal, as in Listing 3.66.

Listing 3.66 Example of illegal statement

```
if <expr> then
    procedure
```

Error 12032 **illegal symbol in declaration**

The declaration contains an illegal or unknown symbol. You have attempted to declare a variable, but its identifier is incorrect or unknown. An example of this error is in Listing 3.67

Fix Possible fixes include:

- Look for a forward routine declaration in the interface part of a unit.

- Look for an illegal token in an assembler routine definition.
- Look for non ANSI-specific symbol use (when the ANS Conformance option in the Pascal Language preference panel is on).

Listing 3.67 Illegal symbol in declaration error

```
unit illegalsymbol;

interface

  procedure a; forward;// cannot have forwarded routine in the
                       // interface part of a unit
implementation

  procedure w; asm;
  label 1;             // cannot have label inside a asm routine
  begin
  end;

end.
```

Error 12019 illegal symbol in factor

You have included an illegal or unknown symbol in your statement, as in Listing 3.68

Fix Look for tokens or operators that are not specific to Pascal.

Listing 3.68 Illegal symbol in factor

```
  x := +-85
  if a==2 then b = 3.0; (* the a==2 is illegal*)
```

Error 12108 illegal usage in this scope

An error message was given because an illegal operation was performed within the scope, as in Listing 3.69.

Listing 3.69 Example of illegal usage in this scope.

```
for i : <expr1> to <expr2> do
    call(i){ where the parameter is by-var }
```

Error 12116 **illegal usage of a selector**

This is an error because `[`, `.`, `^` can only occur with the corresponding types, as in Listing 3.70.

Listing 3.70 Example of illegal usage of a selector.

```
i : integer;
  i^ := 4;
```

Error 12100 **illegal use of inline function**

This assembler error is given because it is illegal to use an inlined routine in this context.

Fix You can't assign an inlined routine to a procedural type.

Listing 3.71 Illegal use of inline function

```
program test;

type proc = procedure;

  procedure a; inline 1234;

var
  p:proc;

begin
  p:=a;
end.
```

Error 12011 **illegal use of keyword**

This is an error because this predefined keyword is not used in the right context.

Fix Use the keyword in a legal context: for example, `cycle`, `leave`, `continue`, and `break` should appear inside a loop block.

Listing 3.72 Illegal use of keyword

```
program test;

  procedure a;
  begin
    cycle;// cycle should be used in a loop
    leave;// leave should be used in a loop
  end;

begin
end.
```

Error 12180 **incorrect syntax for $$Shell(id) substitution**

(MPW only) A warning was issued because the syntax used to specify a path variable is incorrect. The default search paths are used instead to locate the unit.

Error 12120 **'INHERITED' must be used in a method definition**

An error is given because the inherited directive can only occur in a method definition, not in an ordinary routine definition. Listing 3.73 is an example of this error.

Listing 3.73 Example of INHERITED must be used in a method definition.

```
procedure foo;
  begin
    inherited bar;
  end;
```

Error 12152 **INLINE is not supported in PowerPC or Intel**

You tried to use inline opcode routines in PowerPC code. The PowerPC compiler doesn't support this feature.

Listing 3.74 **INLINE is not supported in PowerPC**

```
{$if POWERPC}
program test;

  procedure a; inline;

begin
end.
{$endif}
```

Error 12066 **INTERFACE expected**

The compiler expect to find the `INTERFACE` keyword, but find another symbol.This is an error because the interface part of a `unit` is missing.

Fix Add the `INTERFACE` keyword after the unit name declaration.

internal compiler error

This error message was generated when the compilation halted. The compiler is functioning improperly.

WARNING! If this error is generated please contact Technical Support by sending in a bug report form. Please include as much information as to when and how the error occurred as possible.

Error 12150 **invalid function name or function name expected**

This error is generated because the compiler expected a routine name, or because the routine name specified is invalid.

Fix Give a routine name or make sure the specified routine name is valid.

```
program test;

  function 1_invalid; // invalid name.
  begin
  end;

  function;           // no name specified.
  begin
  end;


begin
end.
```

Error 12149 **invalid procedure name or procedure name expected**

This error is generated because the compiler expected a routine name, or because the routine name specified is invalid.

Fix Give a routine name or make sure the specified name is valid.

Listing 3.75 **Invalid procedure name or procedure name expected**

```
program test;

  procedure 2-bad;{ invalid name }
  begin
  end;

  procedure;      { no name }
  begin
  end;

begin
end.
```

Error 12147 **invalid program name or program name expected**

This error is generated because the compiler expected a program name, or because the program name is invalid.

Fix Possible fixes include:

- Add a program or unit name.
- Make sure the name is valid according to the Pascal syntax.

Listing 3.76 invalid program name or program name expected

```
program;  {missing program name}

begin
end.
```

Error 12148 **invalid unit name or unit name expected**

This error is generated because the compiler expected a unit name or because the unit name specified is invalid. The unit name must be the same as the file name.

Fix Specify a unit name, or make sure the unit name is valid.

Error 12007 **invalid variant record CASE type**

This error is given when the CASE statement selector type in the variant record part is not a scalar type.

Fix Replace the case selector type with a scalar type, as a subrange, enumerated or ordinal type (i.e. boolean, char, a range, integer, etc...).

Listing 3.77 invalid variant record CASE type

```
program test;

  procedure a;
  type
    t_rec = record
```

```
      case CaseSelector: real of {real type is not a scalar type}
        true: ();
    end;
  begin
  end;

begin
end.
```

J to L (Pascal)

These are Pascal compiler error messages that begin with *J*, *K*, or *L*.

Error 12091 **label error**

This error is given because a label is define more than once in a given scope.

Error 12093 **label range error**

An error is given when the case label ranges lower bound is larger than the upper one. Listing 3.78 is an example of this error.

Listing 3.78 **Example of label range error.**

```
case <expr> of
    5..3 :
    end;
```

Error 12134 **local data > 224 bytes**

This PowerPC assembler error occurs because you defined more than 224 bytes of local variables.

Fix You must create a stack frame for this routine, since it has more than 224 bytes of local variables. For more information, see "The Built-In Assembler" in the *Pascal Language Manual*.

Error 12084 **local variables size > 32K**

The total amount of memory used to allocate local variables has exceeded 32KB. This error often occurs when declaring arrays that are too large (Listing 3.79).

Fix Make some of the variables global instead.

Listing 3.79 Local variables size > 32K error

```
PROCEDURE BoomArray();

VAR
  { Error: this local array is
    greater than 32Kbytes }
  wayTooBig : ARRAY [1 .. 100000] OF INTEGER;
```

M to O (Pascal)

These are Pascal compiler error messages that begin with *M*, *N*, or *O*.

Error 12165 **macros cannot be parameters of other macros**

The compiler gives this error because you tried to define a macro with a parameter that is a macro.

Listing 3.80 Macros can't be parameter of other macros

```
program test;

{$definec a 1}
{$definec x(a) a+1}

begin
end.
```

Error 12119 **method not declared in *class name***

The error occured because the underlined method is declared as a member of this class, but wasn't previously defined in the class definition.

Listing 3.81 Method not declared in *class name*

```
program test;

type

  obj = object
    procedure declared;
  end;


  procedure obj.declared;
  begin
  end;

  {Error: method not declared in obj.}
  procedure obj.notdeclared;
  begin
  end;

begin
end.
```

Error 12104 **missing $IFC directive**

This error occured because the compiler encountered a {$ELSEC} or {$ENDC} directive without first finding a matching {$IFC} directive.

Listing 3.82 Missing $IFC directive

```
program test;

{Error: no '$IFC' directive.}
```

```
{$ELSEC}
  procedure a;
  begin
  end;
{$ENDC}

{Error: no '$IFC' directive.}
  procedure b;
  begin
  end;
{$ENDC}

begin
end.
```

Error 12140 **missing array initializer**

This error occurred because the array initializer doesn't contain enough elements. Listing 3.83 is an example which triggers this error.

Listing 3.83 Missing array initializer

```
program test;

type
  arrType = array [1..5] of integer;

const

  x: arrType = (1,2);

begin
end.
```

missing '$IFC' directive

The compiler encountered a {$ELSEC} or {$ENDC} directive without first finding a matching {$IFC} directive.

Error 12193 **must be a local variable**

The compiler gives this error because the underlined loop index variable is not defined in the current scope. Only locally defined variables can be used as a loop index.

Listing 3.84 Must be a local variable

```
program test;

var i:integer;

  procedure a;
  begin
    for i:=1 to 10 do  // variable 'i' is not local to
                       // procedure 'a'
    ;
  end;

begin
end.
```

Error 12183 **must be a pointer or an object**

This is an error because the actual declaration for an anonymous type must be either a pointer or an object.

Error 12046 **must be an array**

This is an error because this variable's type must be an array. Left and right bracket (`[,]`) can be used with set, array and string type variables only.

Fix Possible fixes include:

- Replace the brackets by an appropriate symbol.
- Change the variable type to an array type.

Listing 3.85 Must be an array

```
program test;

  procedure a;
  var
    s:string;
    c:char;
  begin
    s:=c[1];   // should be c:=s[1];
  end;

begin
end.
```

Error 12185 must be an ordinal type

An error because formal discriminants to a schema must be an ordinal type.

Error 12055 must be assignable

The left hand side of an assignment statement can't be assigned a value. Listing 3.86 shows an example.

Fix Replace the left-hand side of the assignment statement by a variable.

Listing 3.86 Must be assignable error

```
CONST
  a = 10;

BEGIN
  a := 20; { Error: constant values can't be reassigned }
...
```

Error 12045 **must be a constant**

A constant expression is expected in a string length specifier, and for initialized variable values.

You tried to use an expression as a constant that is not a constant expression. Listing 3.87 shows an example.

Fix Replace the non-constant identifier by a constant value or a constant identifier.

Listing 3.87 Must be a constant error

```
CONST
  b = 10; { OK }
  x = func();
{ Error: function call is not allowed }
  ...

CASE finalValue OF
  1 : f := 10; { OK }
  b : f := 2;
{ Error: b is not a constant expression }
  ...
```

must be a function

A procedure is called where a function is expected (Listing 3.88).

Listing 3.88 Must be a function error

```
FUNCTION Func() : INTEGER; FORWARD;
...
PROCEDURE Func();
{ Error: expected a function definition }
BEGIN
END;
```

Error 12121 **must be an object type**

The compiler gives this error because this variable's type must be an object type. Listing 3.89 shows an example of this error.

Fix Make sure the variable you're referring to is an object.

Listing 3.89 Must be an object type

```
program test;

type

  rec = record
      a,b: integer;
    end;

  obj = object (rec)
    end;

begin
end.
```

Error 12166 **must be an open array parameter**

This error is given because the `HIGH` function can only be used with an open array; that is, an array declared with no range. An open array can only be declared in routine formal parameter list. Listing 3.90 shows code that triggers this error.

Listing 3.90 Must be an open array

```
program test;

var
  arr: array [1..2] of integer;
  i:integer;

begin
```

```
  i:=high(arr);
end.
```

Error 12048 **must be a pointer**

This is an error because this variable's type must be a pointer.

Fix Every type definition must be declared before it is used in any new type declaration, unless the type definition is a pointer type.

Possible fixes include:

- Declare the type definition before the new type declaration.
- Define the type definition as a pointer type.

Listing 3.91 Must be a pointer

```
program test;

type
  t_ptr = t_rec;  // should be: t_ptr = ^t_rec;
  t_rec = record  // or: t_rec should be a pointer type
    c: char;
  end;

begin
end.
```

must be a procedure

A function is used where a procedure is expected (Listing 3.92).

Listing 3.92 Must be a procedure error

```
PROCEDURE Func() : INTEGER; FORWARD;
...
FUNCTION Func();
{ Error: expected a procedure definition }
```

```
BEGIN
END;
```

Error 12050 **must be a range**

An error was given because this variable's type must be a range. In an array type, the length value should be defined as a range type or a range expression.

Fix Replace the erronous expression by a range type.

Listing 3.93 Example of must be a range error

```
program test;

type
  t_arr = array [integer] of char;

begin
end.
```

Error 12051 **must be a record**

This is an error because this variable's type must be a record. The record type delimiter (.) is used with a non-record type variable.

Fix Possible fixes include:

- Replace the wrong delimiter with the appropriate one.
- Change the variable type to a record type.

Listing 3.94 Must be a record

```
program test;

var
  s: array [0..0] of char;

begin
```

```
  s.0:='a';// should be: s[0]:='a';
end.
```

Error 12053 **must be a scalar**

An error was given because this variable's type must be an ordinal type or an enumeration. Any `FOR` loop control variable and `CASE` constant expression should be of any scalar type.

Fix Possible fixes include:

- `FOR` loop: change the loop control variable type to any scalar type.
- `CASE` statement: change the case statement to multiple `IF` conditions.

Listing 3.95 Must be a scalar

```
program test;

  procedure a;
  var
    r:real;
  begin
    case r of
    end;
  end;

  procedure b;
  var
    r:real;
  begin
    for r:=0.0 to 1.0 do
      ;
  end;

begin
end.
```

Error 12124 **must be a text file**

An error was given because this variable's type must be a file of type 'text'. Listing 3.96 shows source that triggers this message.

Listing 3.96 Must be a text file

```
program test;

var
  f: file of integer;

begin
  {...}
  writeln (f);
end.
```

Error 12054 **must be a variable**

This is an error because this identifier was not declared as a variable. A constant identifier, or any identifier that is not a variable (as a type name or a procedure name), is used where a variable name is expected.

Fix Replace the erroneous identifier by a variable identifier.

Error 12106 **no parameter list**

This ANS Pascal error is given because there is no parameter list specified inside the parenthesis in the routine declaration.

Fix Remove the parenthesis or add a parameter list.

Listing 3.97 No parameter list

```
program test;

{$pragma ANS_conformance on}

  procedure a ();
```

```
  begin
  end;

begin
end.
```

Error 12107 **not in program parameters**

This is an ANS Pascal error message. It is generated when the standard input or ouput files are not declared in the program header and are used in the source code.

Fix Declare the file in the program header.

Listing 3.98 Not in program parameters

```
program test {(output)};

{Error: no program parameter (ANS Pascal only).}

begin
  writeln ('Hello world.');
end.
```

Error 12010 **number is out of range**

This is an error because the literal number is out of the range defined for the variable/field/parameter.

Error 12035 **number overflow**

The compiler is signalling an attempt to assign a numeric constant value to a variable that's greater than the maximum allowed for that data type. Listing 3.99 give an example of this error.

Listing 3.99 Number overflow error

```
VAR
  a : INTEGER;
```

```
BEGIN
  a := 10000000000000000000000000;
{ Error: too big! }
  ...
```

Error 12022 **object cannot contain file component**

This is an error because a file cannot be made out of classes. Your statement contains an invalid use of a file type component.

Fix Possible fixes include:

- Look for a function that returns a text file component.
- Look for an assignment between objects that contain a file component.

Listing 3.100 Object cannot contain file component

```
program test;

  function a : Text; {function cannot return a text file component}
  begin
  end;

  procedure b;
  type
    t_rec = record
      f: file;
    end;
  var
    r1, r2: t_rec;
  begin
    r1 := r2;       {illegal assignment since 't_rec'}
                    {contains a file component}
  end;

begin
end.
```

Error 12043 **object not printable**

This is an error because it's impossible to print the value of this type. This type is not supported by the standard routine. This happens for `StringOf()`, `ReadString()`, `Val()`, `WriteLn()` and `ReadLn()`.

Fix Add your own routine to print, read, or convert the desired variable type.

Listing 3.101 Object not printable

```
program test;

  procedure a;
  var
    enum: (red, blue, green);
  begin
    enum:=red;
    writeln(enum);
  end;

begin
end.
```

Error 12069 **OF expected**

The compiler generates this message because the code is missing the keyword `OF` in either a `CASE` statement, an array, a set or a variant record type.

Fix Add the expected keyword.

Listing 3.102 'OF' expected

```
program test;

type
  t_arr = array [0..0] {of} char;
  t_set = set {of} char;
  t_rec = record
```

```
    case b:boolean {of}
      false: ();
  end;

  procedure a(i:integer);
  begin
    case i {of}
      0: ;
    end;
  end;

begin
end.
```

Error 12006 **out of range**

An error message is given when the value is out of the range defined for the variable, field, or parameter.

This kind of error is detected only if the option **Activate Range Checking** is selected in the pop-up menu Debugging Aids (Pascal Language settings panel). **Activate Range Checking** finds any out of range static constants at compile-time, and generates code for dynamic variables.

Fix Possible fixes include:

- For a static value—Change the value so that it's within the type
limits.
- For a dynamic variable—Change the variable type for any one that can hold every expected values (i.e. replace `integer` by `longint`).

Listing 3.103 Out of range

```
program test;

{$pragma rangecheck on}
```

```
  procedure a;
  type
    t_array = array [1..4] of char;
  var
    a: t_array;
    s: string;
    i: 1..5;
  begin
    a[0]:='0';
    i:=maxint;
    for i:=0 to 5 do
      ;
    s[-1]:='a';
    s[500]:='b';
  end;

begin
end.
```

P to R (Pascal)

These are Pascal compiler error messages that begin with *P, Q,* or *R*.

Error 12024 **parameter mismatch**

The declaration and the definition of your routine do not have the same prototype (Listing 3.104).

Fix Change the parameter list of the invalid prototype to be the same type as the valid one.

Listing 3.104 Parameter mismatch error

```
unit parametermismatch;

interface

  procedure a(i1, i2, i3: integer);
```

```
implementation

  procedure a(i1:integer);// should be:
                         // procedure a(i1, i2, i3: integer);
  begin
  end;

  procedure b(c:char; i:integer); forward;

  procedure b(c:char); // should be: procedure b;
  begin               // or: procedure b(c:char; i:integer);
  end;

end.
```

Error 12027 **parameter missing**

A routine call does not have enough parameters to match the routine's definition (Listing 3.105).

Fix Modify the parameter list of the routine call or its definition.

Listing 3.105 Parameter missing error

```
program test;

  procedure a(c:char);
  begin
  end;

begin
  a;  // should call routine 'a' with one parameter
end.
```

Error 12178 **precompiled unit *unit* can only be used as prefix file**

The compiler gives this error message because a precompiled unit cannot be included in source as a unit with `{$I}` or the USES list.

To uses a precompiled unit, type the prefix file's filename in the **Prefix File** edit field of the **Pascal Language** settings panel.

Listing 3.106 Precompiled unit *unit* can only be used as prefix file

```
program test;

{$I MacOSIntf_68K}
uses MacOSIntf_68K;  //ERROR

begin
end.
```

Error 12088 preprocessor nesting too deep

The compiler gives this error when there are too many nested compilation directives. A maximum of 32 nested levels are allowed.

Fix Reorganize the compilation directives so you don't exceede the maximum number of nested levels allowed (32).

Error 12126 procedural variable can’t get nested routine

The compiler gives this error because you tried to assign a local (nested routine to a procedural type variable.

Error 12031 procedure already FORWARD

The underlined routine is declared as `FORWARD` more than once in the current scope. Listing 3.107 gives and example of this error.

Fix There are two suggested fixes to this error:

- Remove the extra declaration.
- Remove the `FORWARD` keyword if your are actually defining the routine.

Listing 3.107 Procedure already FORWARD error

```
program test;

  procedure a; forward;
  procedure a; forward;   {procedure 'a' is already}
                          {declared as FORWARD}

begin
end.
```

Error 12047 **procedure incorrectly used as a function**

This error occurs when a procedure is used as a function.

Fix Possible fixes include:

- Replace the function call by a procedure call.
- Replace the procedure declaration/definition by a function declaration/definition.

Listing 3.108 Procedure incorrectly used as a function

```
program test;

  procedure a;
  begin
  end;

var
  c: char;

begin
  c:=a();  {ERROR: a is already a procedure call}
end.
```

Error 12109 **program parameter redeclaration**

This ANS Pascal error message is given when the file was already declared as in Listing 3.109.

Listing 3.109 Example of program parameter redeclaration.

```
program (output, output);
```

Error 12012 **range base type mismatch**

The compiler gives this error because the range type has an invalid definition. Listing 3.110 shows an example of this error.

Fix Possible fixes include:

- Reverse the order of the left and right base range if the left base is greater than the right one.
- Decrease the value of the right base if it is too big for any predefined scalar type.

Listing 3.110 Example of range base type mismatch

```
program test;

  procedure a;
  var
    r1: 5..1;                    // left base > right base
    r2: 1..9999999999999999999; // right base > maxlongint
  begin
  end;

begin
end.
```

Error 12039 **redeclaration of routine *routine name***

The compiler gives this error because a function or procedure, named *`routine name`*, is declared more than once in the same scope.

Fix Remove the invalid routine declaration.

Listing 3.111 Redeclaration of routine *routineName*

```
program test;

  procedure a(var c:char);
  begin
    c:='a';
  end;

  procedure a(c:char); //ERROR: duplicate routine name declaration
  begin
    c:='a';
  end;

begin
end.
```

Error 12083 **redundant symbol**

(Warning) The compiler issues this warning because this character is illegal in Pascal.

Fix Remove the illegal character or change to an appropriate one.

Listing 3.112 Redundant symbol

```
program test;

begin
  !   // This is an illegal character in Pascal.
end.
```

Error 12026 **result type mismatch**

The compiler generated this error message because the return type of the functionnal parameter or variable is not the same as the return type of the function used in the assignment.

Fix Change the functionnal type or its definition to be the same.

Listing 3.113 Result type mismatch

```
program test;

  function a : integer;
  begin
    a:=0;
  end;

type
  t_func = function : real;
var
  f: t_func;

begin
  f := a;   {the return type of a() and 'f' are not the same}
end.
```

Error 12044 **routine *routine name* declared but undefined**

(Warning) This warning lets the compiler find all declared routines that were not defined. You did not implement a routine that you declared in the interface or declared `forward`. You need to enable the option **Undefined Routines** in the **Pascal Warnings** settings panel in order to trap the error.

Fix Possible fixes include:

- Define the undeclared routine.
- Add the unit that defines it in the uses clause.

Listing 3.114 Routine *routine name* declared but undefined

```
unit undefroutine;

interface

  procedure a;

implementation
```

```
end.
```

S to T (Pascal)

These are Pascal compiler error messages that begin with *S, T*.

Error 12073 **scalar value expected**

The compiler expects a scalar value, such as a boolean, enumeration, range, char, or integer. For example, in a FOR loop statement, the expression that follows the BY keyword should be a scalar value.

Fix BY is an optional keyword. Remove it, or change its value with a scalar.

Listing 3.115 Scalar value expected

```
program test;

var
  i,j:integer;
begin
  j:=1;
  for i:=0 to 10 by j do   {ERROR}
    ;
end.
```

Error 12118 **segmentation directive must be placed after 'IMPLEMENTATION'**

Segmentation directives, {$S *segmentName*}, cannot be placed in the interface part of a unit. Instead, place segmentation directives in the implementation part.

Fix Place the segmentation directives in the implementation part.

Listing 3.116 Segmentation directive must be placed after 'IMPLEMENTATION'.

```
unit baddirplace;

interface

{$S segment1}

implementation
end.
```

Error 12013 set base type mismatch

This is an error because this type cannot be used as a base type to construct a set, as in Listing 3.117.

Fix Set base type should be of any scalar type within the range of `0..2047`.

Listing 3.117 Example of set base type mismatch.

```
program test;

type
  t_s1 = set of integer;
  t_s2 = set of real;

begin
end.
```

Error 12025 size mismatch for universal parameter

This error is given when UNIV parameters are not the same size. UNIV parameters must match in sizes.

Error 12168 **standard routine cannot be assigned**

You cannot assign a standard routine to a procedure's parameter. A standard routine is one of Pascal's built-in routines, like abs. For example:

Listing 3.118 Assigning a standard routine to a procedure's parameter

```
procedure foo ( function a ( i : integer ) :
       integer );
begin
end;

{ . . . }

foo(abs);{ ERROR: cannot assign the built-in
          function abs to a procedure's
           parameter          }
```

Error 12158 **string mismatch, the parameter could be mangled in the called routine, use a 'const' parameter**

(Warning) The string you're passing to a routine is longer than the routine expects. Since this string parameter is a value parameter, copying the string could corrupt memory.

Fix Use a const parameter.

Listing 3.119 String mismatch, the parameter... error

```
program test;

type
  str10 = string[10];
  str20 = string[20];


  procedure a(x: str10);
```

```
  begin
    writeln(x);
  end;

var
  y: str20;

begin
  a(y);
end.
```

Error 12008 **string too long for assignment**

The string literal assigned to a string variable is longer than the string variable's length (Listing 3.120).

Fix Possible fixes include:

- Increase the length of the string type (to a maximum length of 255 characters).
- Decrease the length of the constant string.

Listing 3.120 String too long for assignment error

```
VAR
  s : STRING[10];

BEGIN
  { Error: s can only be 10 characters long }
  s := 'abcdefghijklmnopqrstuvwxyz';
...
```

Error 12146 **subrange type expected**

You must use a scalar type to construct a subrange type.

Listing 3.121 Subrange type expected

```
program test;
```

```
const
  c1 = 1.1;
  c2 = 2.2;

type
  subRange = c1..c2;

var
  i:c1..c2;
begin
end.
```

Error 12075 **'THEN' expected**

This error occurs when the `THEN` keyword is missing in an `IF` statement (Listing 3.122).

Fix Add the missing `THEN`.

Listing 3.122 'THEN' expected error

```
IF a = 'b' THEN b := a; { OK }
IF x = 0 { Error: no THEN keyword }
  y := 100;
```

Error 12076 **'TO' or 'DOWNTO' expected**

A `TO` or `DOWNTO` keyword is missing in a `FOR` statement (Listing 3.123).

Fix Add the missing `TO` or `DOWNTO` keyword.

Listing 3.123 'TO' or 'DOWNTO' expected error

```
FOR j := 1 TO 10 { OK }
  k := k * 2;

FOR i := 0 DO { Error: no TO or DOWNTO clause }
```

Error 12141 **too many array initializers**

This array initializer contains more elements than the array can accept.

Fix Remove all extra array elements.

Listing 3.124 Too many array initializers

```
program test;

type
  arrType = array [1..5] of integer;

const
  x: arrType = (1,2,3,4,5,6);

begin
end.
```

Error 12131 **too many include files**

This error message is displayed when the number of include files exceeds the capacity of the compiler.

Error 12130 **too many nested directives**

This error message is generated when the compiler encounters too many levels of nesting.

Listing 3.125 Too many nested directives error (example)

```
program test;

{$push}
{$pop}
{$pop}
```

```
begin
end.
```

Error 12162 **too many nested macros**

You exceeded the maximum nesting level for a macros. You cannot nest them more than 50 deep.

Error 12129 **too many opcodes for inline routine**

The compiler gives this error because you have specified more than 512 opcodes for this inlined routine.

Fix Break the routine into two routines, each with no more than 512 opcodes.

Error 12077 **type expected**

A data or object type is missing in a variable declaration (Listing 3.126).

Fix Add a type identifier.

Listing 3.126 **Type expected error**

```
VAR
  a : CHAR; {OK }
  i : ;  { Error: missing type }
```

Error 12153 **type *identifier* had a forward declaration, the compiler cannot change it to *identifier***

You didn't define the type, and the compiler cannot change a variable of that type to a new type. To allow this type coercion, turn on the **Relax Pointer Compatibility** option in the Pascal Language settings panel.

Error 12040 **type mismatch**

There is a type mismatch between the destination variable or parameter and the assignment or the call.

Also, the type of an item in an expression is not compatible with the other types used in the expression. An example of this error is in Listing 3.127.

Fix Possible fixes include:

- Add a type cast (if necessary).
- Change the variable or parameter type.

Listing 3.127 Type mismatch error

```
  a := 'A';
  b := 'a' - 'A';
{ Error: characters aren't numeric }
```

U to Z (Pascal)

These are Pascal compiler error messages that begin with *U*, *V*, *W*, *X*, *Y*, or *Z*.

Error 12081 **unclosed comment**

A comment has not been terminated (Listing 3.128).

Listing 3.128 Unclosed comment error

```
{ Error: unclosed comment error }
{$I test.p
VAR
  i, j : INTEGER;
...
```

Error 12002 **undeclared identifier**

A variable or constant name is used in source code, but has not been declared.

Fix Possible fixes include:

- Declare the identifier.
- If the identifier is declared elsewhere, add that unit in the uses clause.
- Look for type mismatch, or misspelling of the identifier name.

Listing 3.129 Undeclared identifier

```
program test;

  procedure a;
  begin
    i:=0;        // variable i is not defined
  end;

  procedure b;
  const
    null = 0;
  begin
    writeln(nul);// type mismatch of the identifier name
  end;

  procedure c;
  begin
    DoNothing;   // DoNothing() is declared in unit Toto.p
  end;

begin
end.
```

Error 12003 **undefined identifier**

The identifier is not defined in this scope.

Fix This error often occurs because a record field, an object field, or a loop variable is undefined.

Possible fixes include:

- Define the identifier.
- Look for type mismatch or misspelling of the identifier name.

Listing 3.130 Undefined identifier

```
program test;

type
  t_rec = record
    achar: char;
    anint: integer;
  end;
var
  rec: t_rec;
begin
  rec.areal:=0.0;// record field is undefined
  rec.aint:=0;// type mismatch of the record field identifier name

  for i:=1 to 5 do  // loop variable not defined
                    // (loop variables should be locals)
    ;
end.
```

Error 12092 **undefined label**

A label has been declared or referenced in a `GOTO` statement but is not used in the source code (Listing 3.131).

Listing 3.131 Undefined label error

```
LABEL 1000;

VAR
  i, j : INTEGER;
```

```
BEGIN
  i := 10;
  IF i > 5 THEN GOTO 1000
      { Error: 1000 never used }
END;
```

Error 12004 **undefined label *labelname***

An error occured when a label was declared but is not defined in it's scope.

Fix Remove the label declaration and recompile. If this label was referenced by a `GOTO` statement, the compiler will give you an error. This way, you know if this label can be removed or must be defined.

Listing 3.132 Undefined label *labelname*

```
program test;

label
  1, _end;

begin
  _end:
  // label 1 is undefined
end.
```

Error 12179 **undefined MPW shell variable**

(MPW only, warning) A warning was issued because the path variable is undefined. The default search paths are used instead to locate the unit.

Error 12005 **undefined pointer *pointer name***

An error occurred when this identifier was assumed to be a pointer but is still undefined.

Fix Possible fixes include:

- Define *pointer name;*
- Add the unit which defines *pointer name*.

Listing 3.133 Undefined pointer *pointer name*

```
program test;

type
  t_pchar = ^t_char;  // t_char is undefined

begin
end.
```

Error 12097 **unexpected end of file**

The compiler reached the end of a source code file before it could read the terminating `END` statement.

Fix Add the terminating `END` statement in the source code file.

Error 12135 **unknown PowerPC instruction mnemonic**

This PowerPC assembler error is given because this mnemonic doesn't exist in the PowerPC instruction set.

Fix Check for a misspelled mnemonic.

Listing 3.134 Unknown PowerPC instruction mneomonic

```
program test;


  procedure a; asm;
  begin
    unknownMnemonic r1,r2;
  end;
```

```
begin
end.
```

Error 12078 **UNIT expected**

The compiler expect to find PROGRAM or UNIT keyword at the begining of your source file. Every source code file must begin with either the `PROGRAM` or `UNIT` keyword.

Fix Add `PROGRAM` or `UNIT` keyword to the beginning of your source file.

Error 12034 **unit *unit* wasn't compiled**

This error indicates that the unit being referenced cannot be compiled because of error(s). The unit being referenced in the project cannot be found within the project. This error often occurs when a `UNIT`'s name and its `filename` don't match. By default, Metrowerks Pascal requires that the name in a unit's `UNIT` statement and the unit's filename (without the filename extension) be the same.

Fix Add the unit or the library that contains the unit to the project, or change the unit's filename to match the name used in the unit's `UNIT` statement.

Error 12175 **unrecognized pragma**

(Warning) This warning is generated when you try to use a pragma that is not valid for the platform target. In order to have this message, you must set the pragma `warn_illpragma` to ON in your code.

Fix Possible fixes include:

- Make sure there is no syntax error in the pragma name.
- Make sure the pragma you try to use is valid for the current target.

Listing 3.135 Unrecognized pragma

```
program test;

{$pragma warn_illpragma on}

{$pragma NoGood} //ERROR - unrecognized pragma

begin
end.
```

Error 12079 **UNTIL expected**

This `REPEAT` statement is missing its `UNTIL` clause. Every `REPEAT` statement must end with an `UNTIL` clause.

Fix Add an `UNTIL` clause at the proper location.

Listing 3.136 'UNTIL' expected

```
program test;

var
  i: integer;

begin
  i := 0;
  repeat
    i := 1+1;
  // UNTIL clause expected here.
end.
```

Error 12123 **unresolved forward class reference to *class***

This error occurred because this class was declared having a forward definition, but the class is still undefined.

Listing 3.137 Unresolved forward class reference

```
program test;

type
  obj = object; forward;

begin
end.
```

Error 12142 **unresolved external class reference to *identifier***

The compiler cannot create an instance of an external class since it doesn't know the class's size.

Listing 3.138 Unresolved external class reference to *identifier*

```
program test;

type
  obj = object; external;

var
  x: obj;

begin
  new (x);
end.
```

Error 12122 **unsafe object reference**

(Warning) This warning is issued when a reference to an object field is unsafe. The Memory Manager may move the object. You must use the pragma `objcheck` to receive this warning.

Listing 3.139 Unsafe object reference

```
program test;

type
```

```
  obj = object
      a,b:integer;
    end;

var
  o: obj;

begin
{$pragmac objcheck on}
  with o.a do begin
  end;
end.
```

Error 12082 **unterminated string**

A string literal doesn't have an ending quote (`'`). Listing 3.140 shows an example of this error.

Fix Terminate the string with an ending quote character (`'`).

Listing 3.140 Unterminated string error

```
program test;

const
  aString = 'This is a string with no ending quote;

begin
end.
```

Error 12087 **unused variable**

The compiler is warning that a variable has been declared, but it is never used in its scope (Listing 3.141). To make this warning active, select **Unused Variables** in the Pascal Warnings settings panel.

Listing 3.141 Unused variable warning

```
FUNCTION ChunkCount(a : INTEGER) : INTEGER;

VAR
  SockStr : STRING[100]; { Warning: SockStr never used }

BEGIN
  ChunkCount := a * 10
END;
```

Error 12101 **value is not stored in register**

The compiler cannot use this value because it is not stored in a register.

Error 12111 **variable *variable* is a loop index**

This ANS Pascal error message is given when you try to use an loop index in a way so it can be modified.

Fix Use a temporary variable instead of the loop index.

Error 12190 **variable identifiers not allowed in expression**

This error occurred because index range expressions cannot contain references to variable identifiers. Index range expressions can only contain references to constant expressions or function calls.

Listing 3.142 Variable identifiers not allowed in expression

```
program test;

type
  schemaType(i:integer) = array[1..i] of integer; // ERROR

var
  j:integer;
  y:schemaType(j);
```

```
begin
end.
```

Error 12089 **variable used but not initialized**

(Warning) A variable is used in an expression without first being assigned a value (Listing 3.143).

Fix Assign a value to the variable before using it in an expression.

Listing 3.143 Variable used but not initialized warning

```
VAR
  i, j, k : INTEGER;

BEGIN
  j := i * 10; { Error: i isn't initialized yet }
  IF (Fib(i) > 100 THEN
    { Error: i isn't initialized yet }
    k := 1;
END.
```

4

Java Error Messages

This chapter gives an alphabetical list of the compiler errors which may be encountered while using Metrowerks Java compiler.

Java Compiler Errors

In this list, errors with variable initial text (such as a class or function name) come first. Errors beginning with a non-alphabetic symbol character come next. After that, errors are listed alphabetically.

Symbol Names (Java)

These are Java compiler error messages that begin with a symbol name, such as the name of a method, variable, or class.

Error 14195 ***class*** **in throws clause must be a subclass of class java.lang.Throwable.**

Any class in the `throws` clause must be a subclass of `Throwable`. Generally, the exception classes you create will be subclasses of `Exception`, which is in turn a subclass of `Throwable`.

Listing 4.1 Throwing a class that is not a subclass of Throwable

```
public class FirstException {
  // Not a subclass of anything.
  // . . .
}

public class SecondException extends Exception {
```

```
  // A subclass of Exception, which is a subclass of Throwable.
  // . . .
}

public class ThrowTestClass {
  public static void c(int x)
      throws FirstException, SecondException {
          // ERROR: FirstException is not a subclass of
          //      java.lang.Throwable.

    // . . .
  }
  // . . .
}
```

Error 14099 ***class*** **is an abstract class. It can't be instantiated.**

You tried to create an instance of an abstract class.

Fix Either declare the class not to be abstract, or create a subclass of the abstract class and then create an instance of the subclass.

Listing 4.2 Instantiating an abstract class

```
public abstract class Animal { /* . . . */ }
public class Bird extends Animal { /* . . . */ }

public class AbstractExample {

  public static void main(String args[]) {
    Animal A = new Animal();
        // ERROR: Animal is an abstract class.
    Bird B = new Bird();
        // OK: Bird is not abstract.
  }
}
```

Error 14175 ***className* is defined in *fileName*. Because it is used outside of its source file, it should be defined in a file called *className.java*.**

This error will only occur when the option Strict File Names is enabled in the Java Language settings panel, and you have not defined a classname is defined in *fileName*. Because it is used outside of its source file, it should be defined in a file called *className.java*.

Error 14092 ***interface* is an interface. It can't be instantiated.**

You tried to create an instance of an interface.

Fix Either change the interface to a class, or create a class that implements the interface and create an instance of the class.

Listing 4.3 Instantiating an interface

```
public interface CanFly { /* . . . */ }
public class Plane implements CanFly { /* . . . */}

public class InterfaceExample {
  public static void main(String args[]) {
    CanFly C = new CanFly(); // ERROR: CanFly is an interface.
    Plane P = new Plane(); // OK: Plane is a class
  }
}
```

Error 14091 ***class1* must be declared abstract and not final. It does not define *method* from *class2*.**

If a class contains abstract methods, you cannot declare it as `final`. Instead, you must declare it `abstract`.

Fix You may have subclassed the class from an abstract class and forgotten to define all the abstract methods. Or you may have declared abstract methods right in the class and accidentally declared the class as `final` instead of `abstract`. Either declare the class to be `abstract`, or define the abstract methods.

Listing 4.4 Declaring an abstract class as final

```
public abstract class Animal {
  abstract void Noise();
  abstract int NumLegs();
}

public final class Bird extends Animal {
  void Noise() { System.out.println( "Tweet!" ); }
  // ERROR: Bird must define NumLegs() to be a final class
}

public final class Shape {
  abstract double Circumference();
  abstract double Area();
  // ERROR: Shape must be declared abstract
  //      since it declares abstract methods.
}
```

Error 14090 ***class1* must be declared abstract. It does not define *method* from *class2*.**

If a class contains abstract methods, you must declare it `abstract`.

Fix You may have subclassed the class from an abstract class and forgotten to define all the abstract methods. Or you may have declared abstract methods right in the class and forgotten to declare it as `abstract`. Either declare the class to be `abstract`, or define the abstract methods.

Listing 4.5 Declaring an abstract class as final

```
public abstract class Animal {
  abstract void Noise();
  abstract int NumLegs();
}

public class Bird extends Animal {
  void Noise() { System.out.println( "Tweet!" ); }
  // ERROR: Bird must define NumLegs() or be declared abstract
```

```
}

public class Shape {
  abstract double Circumference();
  abstract double Area();
  // ERROR: Shape must be declared abstract
  //      since it declares abstract methods.
}
```

Error 14013 ***symbol* expected.**

This error will be signaled when the compiler expects to find a certain keyword or punctuation, as shown in Listing 4.6.

Listing 4.6 *symbol* expected

```
if i = 3)  //  will give "( expected"
return;
```

Error 14153 ***symbol* must be an interface.**

Where the compiler expected to see the name of an interface, you used the name of another symbol. For example, you may have used the name of a class instead of an interface in an `implements` clause.

Listing 4.7 Using a class name where an interface name is expected

```
public abstract class Animal { /* . . . */ }
public class Bird implements Animal { /* . . . */ }
  // ERROR: In an implements clause, you must use the name of an
  //      interface, not a class.
public class Bear extends Animal { /* . . . */ }
  // OK
```

Error 14114 ***symbol* not supported.**

This error is a "convenience" error. It is signaled when certain unsupported keywords or expressions (`const, goto, placement new`) from C++ are seen by the compiler.

Punctuation Marks (Java)

These are Java compiler error messages that begin with punctuation marks.

Error 14028 **[] can only be applied to arrays. It can't be applied to *type*.**

You applied `[]` to `type`, which is not an array.

Fix You may have misdeclared the variable, or used too many dimensions in an array reference.

Listing 4.8 Using [] with an integer

```
int x;
int[] y = new int[3];
int[][] z = new int[3][3];

x[2] = 1; // ERROR: x is an integer
y[2] = 2; // OK
y[2][2] = 3;// ERROR: y is a 1-dimensional array
z[2][2] = 4;// OK
```

A to B (Java)

These are error messages that begin with *A* or *B*.

Error 14103 **a "break" or "continue" must transfer control within the same method.**

Methods in block local classes must not "break" or "continue" to the enclosing method. An illegal break is shown in Listing 4.9.

Listing 4.9 Illegal break

```
public class foo {

  public void method( int x) {
```

```
    switch (x) {

      case 1:
      {
        Object o = new Object() {

          public String toString() {

            //...
            break; //illegal; must not break out of method scope
          }
        }
      }
    }
  }
}
```

Error 14254 **A declaration cannot be labeled: *labelName***

The compiler generates an error when a declaration is labelled, as shown in the example below.

```
class A {
 {
   label: int i =  3;
 }
}
```

Error 14263 **A default constructor cannot be created for this class because the constructor of its superclass throws the exception *exceptionType*. Define the constructor for this class explicitly.**

This error is explained in the code example below.

```
class Parent {
  Parent() throws Exception { }
}

class Child {
    //When the compiler attempts to create the default constructor
```

```
    //here, it fails because the default super() call throws a
    //checked exception.  Instead, declare the Child default
    //constructor explicitly.
}
```

Error 14253 **A statement cannot be labeled with the same identifier as one of its enclosing statements: *labelName***

If duplicate label names appear accross accesible scopes, this error occurs. For example:

```
class A {

  {
    label: while (true) {

      label: while (true) {
        break;
      }
      break;
    }
  }
 }
```

Error 14133 **Abstract and native methods can't have a body: *method***

You declared a method to be abstract or native, but you gave the method a body. Abstract methods are defined in the subclass's body. Native methods are defined in a source file written with another language, such as C.

Listing 4.10 Abstract and native methods with bodies

```
public abstract class Animal {
  abstract void Noise();
  abstract int NumLegs() { return 0; }
    // ERROR: An abstract method can't have a body.
}

public class Bird extends Animal {
```

```
  void Noise() { System.out.println( "Tweet!" ); }
  native int NumLegs() { return 2; }
    // ERROR: A native method can't have a body.
}
```

Error 14217 **Abstract methods can't be final: *method***

This error is given when a method is declared to be both abstract and final.

Error 14218 **Abstract methods can't be native:*method***

This error is given when a method is declared to be both abstract and native.

Error 14215 **Abstract methods can't be private: *method***

You cannot declare a method to be both private and abstract. A private method cannot be overridden in a subclass. An abstract method must be overridden in a subclass to be useful.

Error 14216 **Abstract methods can't be static: *method***

This error is given when a method is declared to be both abstract and static.

Error 14219 **Abstract methods can't be synchronized: *method***

This error is given when a method is declared to be both abstract and synchronized.

Error 14052 **Access across scopes to the private member *targetMemberName* in *className* is not implemented. The reference will succeed if the member is given package scope.**

The compiler generates an error when you attempt to access across scopes to the private member.

Error 14061 **Ambiguous class: *symbol* and *symbol***

If a class could be resolved to two different imported packages, this error is thrown.

```
import java.util.*;
import java.sql.*;

class aClass {

  Date aDate; //Could be "java.util.Date" or "java.sql.Date"


}
```

Error 14194 **Ambiguous name: *typeName* is both a class and a package.**

If a qualifier resolves to both a package and a class, this error is generated.

Listing 4.11 Qualifier resolves to both a package and a class

```
package foo;

class foo {

  class feem {

  }
}

class feem {

  public void aMethod() {

    feem aFeem = new foo.feem(); //ERROR: which feem is intended?
  }
}
```

Error 14201 **An error has occurred in the compiler; please file a bug report (support@metrowerks.com) using the e-mail bug report form in the Release Notes folder.**

There is a bug in the compiler that caused it to raise a fatal error that it could not handle. Please file a bug report to Metrowerks Technical Support at `support@metrowerks.com` with the report form on your CodeWarrior CD. Be sure to include code which triggers this error.

Error 14202 **An exception has occurred in the compiler; please file a bug report (support@metrowerks.com) using the email bug report form in the Release Notes folder.**

There is a bug in the compiler that caused it to raise an exception it could not handle. Please file a bug report to Metrowerks Technical Support at `support@metrowerks.com` with the report form on your CodeWarrior CD. Be sure to include code which triggers this error.

Error 14155 **An interface can't implement anything; it can only extend other interfaces.**

You declared an interface and used the `implements` keyword where you should have used the `extend` keyword.

Fix When you list the superinterfaces of an interface, use the `extend` keyword.

Listing 4.12 Using implements, instead of extends

```
public interface CanFly { /* . . . */ }

public interface CanFlyInSpace implements CanFly { /* . . . */ }
  // ERROR: Use extends, not implements

public interface CanBeSuperSonic extends CanFly { /* . . . */ }
  // OK
```

Error 14168 **Argument can't have type void: *symbol***

You declared a method's argument to be `void`. Only methods themselves can be `void`.

Listing 4.13 Declaring an argument to be void

```
int X( void a ) {
  // ERROR: An argument can't be void.
  // . . .
}
```

Error 14172 **Arithmetic exception.**

You tried to perform an illegal arithmetic operation, such as dividing by zero.

Listing 4.14 Dividing by zero

```
int i = 1 / 0; // ERROR: Cannot divide by zero.
```

Error 14031 **Array constants can only be used in initializers.**

You can use array constants only in a variable's declaration to initialize the array.

Listing 4.15 Using array constants

```
int x[] = { 1, 2, 3 };// OK

int y[] = new int[3];
y = { 1, 2, 3 }   // ERROR

int z[] = new int[3];
z[1] = 1; z[2] = 2; z[3] = 3; // OK
```

Error 14098 **Array dimension missing.**

You forgot to use a dimension when creating a new array.

Listing 4.16 Forgetting the array dimension

```
int[] a = new int[]; // ERROR: Need a dimension.
int[] b = new int[3];// OK
```

Error 14027 **Array index required.**

You forgot to use an index when referencing an element in an array.

Listing 4.17 Forgetting the array index.

```
int z[] = new int[3];
z[] = 1; // ERROR: Need an index.
z[2] = 2;// OK
```

Error 14259 **Array of void is not a valid type.**

The degenerate type `void[]` is illegal. For example,

```
void[] anArray; //error
```

Error 14246 **Assignment expressions are not permitted as the third subexpression of a conditional expression (that is, a ? : expression). Try parenthesizing this assignment expression.**

Expressions of the following type, where an assignment is used in the last subexpression of a conditional expression, are illegal:

```
int i = ((x == 3) ? 4 : x = 5);
```

Error 14049 **Attempt to assign a blank final *variableName* variable in a loop. The initialization must occur exactly once.**

Blank finals are final variables which have no initialization expression in their declaration. These variables must be definitely assigned once and only once, and their initialization cannot occur in a looping construct.

Listing 4.18 Attempting to assign a blank final in a loop

```
class aClass {


  aClass() {

    final int i; //blank final

    for (int x = 0; x < 3; x++)
      i = 3; //blank final is assigned inside the for loop
  }
}
```

Error 14050 **Attempt to assign to a variable *variableName* in a different method. From enclosing blocks, only final local variables are available.**

Methods in block local classes can only assign to blank local variables from the enclosing method.

Listing 4.19 Attempt to assign to a variable *variableName* in a different method.

```
public class outerClass {
  public void outerMethod (int x) {
    final int y;

    new Object() {
      public int hashCode() {
        x++;   //not allowed; x is not final
        y = 3; //OK
      }
    }
  }
}
```

Error 14039 **Attempt to reference field *field* in a *variable*.**

You tried to access a field in a variable that isn't a class and has no fields.

Listing 4.20 Referencing a field in a variable with no fields

```
int x;
System.out.println (x.length);
  // ERROR: x is an int and has no fields.
```

Error 14046 **Attempt to reference method *method* in *variable* as an instance variable.**

You tried to access *method* as though it were an instance variable in *variable*.

Fix When you call a method with no arguments, you must always follow the method's name with an empty pair of parentheses. Either change the declaration to be a instance variable, or add an argument list to the method's call.

Listing 4.21 Using a method as an instance variable

```
public class Bird {
  void Noise() { System.out.println( "Tweet!" ); }
  int NumLegs() { return 2; }
}

public class AbstractExample {
  public static void main(String args[]) {

    System.out.println (B.NumLegs);
      // ERROR: NumLegs is a method, not an instance variable
  }
}
```

Error 14051 **Attempt to use a non-final variable *variableName* from a different method. From enclosing blocks, only final local variables are available.**

Methods in block local classes can access local variables from the enclosing method, but only if these variables are final.

Listing 4.22 Attempting to use a non-final variable from a different method.

```
public class outerClass {

  public void outerMethod (int x) {

    final int copyOfx = x;

    new Object() {
      public String toString() {
        return "x is " + x + " and copyOfx is " + y;
        //reference to copyOfx is allowed, reference to x is not.
      }
    }
  }
}
```

Error 14059 **Blank final variable *variableName* may not have been initialized. It must be assigned a value in an initializer, or in every constructor.**

The compiler must be able to determine that all blank final variables are definitely assigned.

Error 14101 **'break' must be in loop or switch.**

A `break` statement can appear only within a `switch`, `while`, `do`, or `for` statement block.

Fix Either make sure your braces are balanced correctly, or rewrite your code to avoid the `break` statement.

C (Java)

These are error messages that begin with *C*.

Error 14144 **Can't access symbol. Class or interface must be public, in same package, or an accessible member class.**

If an attempt is made to use an inaccessible class, this error is signaled.

An example of this error is shown in Listing 4.23.

Listing 4.23 **Can't access symbol. Class or interface must be public, in same package, or an accessible member class.**

```
import java.util.*;

HashtableEntry[] entry = new HashtableEntry[]; //illegal;
//HashtableEntry is only accessible within the java.util package
```

Error 14072 **Can't access protected field *variable* in *class*. *class* is not a subclass of the current class.**

You declared a variable to be `protected` or `private protected` and are accessing it from a class that doesn't have access to it. If you declare a class's variable to be `protected`, you can access it only in a subclass of that class or in other classes in the same package. If you declare a class's variable to be `private protected`, you can access it only in a subclass of that class.

For example, say you have this class declaration in the package `foo`:

Listing 4.24 **A class with a protected variable.**

```
package foo;

public class A {
  protected int m;
```

```
  public int n;
}
```

This code won't compile:

Listing 4.25 Accessing a protected variable

```
package bar;
import foo.*;

public class X extends A {
  public int sum(A a) {
    return a.m + a.n;
    // ERROR: a.m is protected and X is a subclass of A, but you
    //     aren't using X's copy of m. You're using another
    //     instance's copy of m.
  }

  public int sum() {
    return m + n;
    // OK: Now you're using X's copy of m.
  }
}
// . . .
```

Error 14073 **Can't access protected inner type *innerClassName* in *targetClassName*. *currentClassName* is not a subclass of the current class.**

Inner classes obey the same access rules as other class members.

Listing 4.26 Can't access protected inner type because the current class name is not a subclass of the current class

```
class A {

  protected static class B {

  }
```

```
}

class C {

  void method() {
    new A.B(); //illegal; B is protected,
               //and C is not a subclass of A.
  }
}
```

Error 14071 **Can't access protected method *method* in *class*. *class* is not a subclass of the current class.**

You declared a method to be `protected` or `private protected` and are accessing it from a class that doesn't have access to it. If you declare a class's method to be `protected`, you can access it only in a subclass of that class or in other classes in the same package. If you declare a class's method to be `private protected`, you can access it only in a subclass of that class.

For example, say you have this class declaration in the package `foo`:

Listing 4.27 A class with a protected variable.

```
package foo;

public class A {
  protected int m() { return 1; }
  public int n { return 2; }
}
```

This code won't compile:

Listing 4.28 Accessing a protected variable

```
package bar;
import foo.*;

public class X extends A {
```

```
  public int sum(A a) {
    return a.m() + a.n();
    //ERROR: a.m() is protected and X is a subclass of A, but you
    //     aren't using X's copy of m(). You're using another
    //     instance's copy of m().
  }

  public int sum() {
    return m() + n();
    // OK: Now you're using X's copy of m().
  }
}
// . . .
```

Error 14048 **Can't assign a second value to a blank final variable: *variableName***

Blank finals are final variables which have no initialization expression in their declaration. These variables must be assigned once and only once.

Listing 4.29 Assigning a blank variable more than once

```
  public class aClass {
    public static final int x;

    static {
      x = 3;
      x = 4; // error here
    }
  }
```

Error 14047 **Can't assign a value to a final variable: *variable***

You tried to change the value of the variable *variable* which was declared `final`.

Fix Either change the variable's declaration, or declare a new variable to hold the value.

Listing 4.30 Assigning a value to a final variable

```
final int SIZE = 512;

int setSIZE(int x) {
  SIZE = x; // ERROR: SIZE is final
  return SIZE;
}
```

Error 14197 **Can't catch *class*; it must be a subclass of class java.lang.Throwable.**

Any class in the `catch` clause must be a subclass of `Throwable`. Generally, the exception classes you create will be subclasses of `Exception`, which is in turn a subclass of `Throwable`.

Listing 4.31 Catching a class that is not a subclass of Throwable

```
public class E1 { /* . . . */ }
public class E2 extends Exception { /* . . .* }
public class CatchTestClass {
  public static void c(int x) {
    try {
      // . . .
    } catch (E1 e1) {
        // ERROR: E1 is not a subclass of Throwable.
        // . . .
    } catch (E2 e2){
        // OK: E2 is a subclass of Exception,
        //   which in turn is a subclass of Throwable.
        // . . .
    }
  }
}
```

Error 14093 **Can't directly invoke abstract method *method* in *class*.**

You tried to directly call a method that was declared to be abstract.

Fix Either remove the method call or define the method.

```
public abstract class Animal {
  abstract int NumLegs();
}
public class Bird extends Animal {
  int NumLegs() { return 2 + super.NumLegs(); }
    // ERROR: Can't invoke super.NumLegs(), since it's abstract.
}
```

Error 14067 **Can't invoke a method on a *symbol*.**

You tried to invoke a method on a variable that isn't a class and has no methods.

Listing 4.32 Invoking a method on an int

```
int x;
x.length();// ERROR
```

Error 14044 **Can't make a static reference to inner class *innerClassName*.**

Inner classes which are not declared static must be referenced through an instance of the outer class.

Listing 4.33 Failure to make a static reference to inner class

```
public class outerClass {

  public class innerClass {

    public void aMethod() { }
  }
}

public class anotherClass {

  public static void anotherMethod() {

    outerClass.innerClass.aMethod();
     //this fails, because no instance of outerClass present
```

```
  }
}
```

Error 14043 **Can't make a static reference to nonstatic variable *variable* in *class*.**

You accessed *variable* in *class* as though it were a static variable, but it actually is an instance variable.

Fix Either access the variable through an instance of the class, or declare the variable to be static.

Listing 4.34 Accessing an instance variable as a static variable

```
public class Bird  {
  int size;
  static int numlegs;
}

public class StaticExample {
  public static void main(String args[]) {
    Bird b = new Bird();

    Bird.size = 12; // ERROR: size is not static.
    Bird.numlegs = 2; // OK
    b.size = 12;// OK
  }
}
```

Error 14097 **Can't make forward reference to *variable* in *class*.**

While initializing a variable, you made a forward reference to a variable that hasn't been defined yet.

Fix Either change the order in which you declare the variables, or re-write the initialization.

Listing 4.35 Illegal forward reference

```
int a = b; // ERROR: b isn't defined yet
int b = 1; // OK
int c = b; // OK
int d = d + 1; // ERROR: Cannot refer to d in
               //     its own initialization
```

Error 14070 **Can't make static reference to method *method* in *class*.**

You accessed *method* in *class* as though it were a static method, but it actually is an instance method.

Fix Either access the method through an instance of the class, or declare the method to be static.

Listing 4.36 Accessing an instance method as a static method

```
public class Bird  {
  static void PrintNumBirds() { /* . . . */ }
  void Fly() { /* . . . */ }
}

public class StaticExample {
  public static void main(String args[]) {
    Bird b = new Bird();

    Bird.Fly(); // ERROR: Fly() is not static.
    Bird.PrintNumBirds(); // OK
    b.Fly();      // OK
  }
}
```

Error 14199 **Can't read: *symbol***

If the compiler has problems getting the text for a source from the IDE, the Java compiler will report this error.

Error 14060 **Can't reference *symbol* before the superclass constructor has been called.**

In an explicit constructor call, either you referred to one of the object's instance variables or instance methods, or you used `this` or `super` in one of the arguments.

Fix Replace the instance variables or instance methods with other values.

Listing 4.37 Referring to instance variables in explicit constructor call

```
class Point {
  int x, y;
  int oldX = 0, oldY = 0;
  static final int DEFAULTX = 0, DEFAULTY = 0;
  Point(int x, int y) { this.x = x; this.y = y; }

  Point() { this(oldX, oldY); }
    // ERROR: oldX and oldY are instance variables.
  Point() { this(DEFAULTX, DEFAULTY); }
    // OK: DEFAULTX and DEFAULTY are static variables.
  Point() { this(this.DEFAULTX, this.DEFAULTY); }
    // ERROR: Cannot use "this" in explicit constructor call.
  Point() { this(0, 0); }
    // OK
}
```

Error 14029 **Can't specify array dimension in a declaration.**

You specified the array's dimension in its declaration.

Fix Specify the array's dimension in the initial value for the array, as shown in the example below. If you're declaring an argument, leave out the array dimension altogether.

Listing 4.38 Specifying an array's dimension in its declaration

```
int a[3] = new int[];// ERROR
int b[]  = new int[3]; // OK
```

```
int c[3] = { 1, 2, 3 };// ERROR
int d[]  = { 1, 2, 3 };// OK
```

Error 14030 **Can't specify array dimension in a type expression.**

In a type expression for an array type, you included the array's dimension. Type expressions are most commonly used when you make an explicit cast.

Fix Remove the dimension.

Listing 4.39 Casting arrays

```
class Animal { /* . . . */ }
class Bird extends Animal { /* . . . */ }

public class ArrayCastTest {
  public static void main(String args[]) {

    Animal animals[] = new Bird[3];
    Bird birds1[] = (Bird[3])animals;// ERROR
    Bird birds2[] = (Bird[])animals; // OK
  }
}
```

Error 14147 **Can't subclass final classes: *class***

You tried to make a subclass of *class*, which is declared final.

Fix Either make a subclass of a different class, or remove the `final` modifier from *class*'s definition.

Listing 4.40 Subclassing a final class

```
final class Bird { /* . . . */ }
class Eagle extends Bird { /* . . . */ } // ERROR
```

Error 14148 **Can't subclass interfaces: *symbol***

You tried to subclass the interface *interface*. A class doesn't extend an interface, it implements an interface.

Fix Either change the class's declaration so that it uses the `implements` keyword instead of the `extends` keyword, or change the interface to a class.

Listing 4.41 Subclassing an interface

```
interface CanFly { /* . . . */ }
class Bird extends CanFly {/*... */}// ERROR
class Plane implements CanFly {/*...*/} // OK
```

Error 14196 **Can't throw *symbol*; it must be a subclass of class java.lang.Throwable.**

Any class in a `throw` clause must be a subclass of `Throwable`. Generally, the exception classes you create will be subclasses of `Exception`, which is in turn a subclass of `Throwable`.

Listing 4.42 Catching a class that is not a subclass of Throwable

```
public class E1 { /* . . . */ }
public class E2 extends Exception { /* . . .*/ }

public class CatchTestClass {
  public static void c(int x) throws E1, E2 {
      // ERROR: E1 is not a subclass of Throwable

    if (x==0)
      throw new E1();// ERROR: E1 is not a subclass of Throwable
    else if (x<0)
      throw new E2();// OK: E2 is a subclass of Throwable
    else {
      // . . .
    }
```

```
  }
}
```

Error 14111 **Case label *symbol* too large for 'switch' on *type***

In a switch statement block, the case label *symbol* is larger than the maximum allowed value for the switch variable of type *type*.

```
short c = 0;
switch (c) {
  case 1:      // OK
  case 100:    // OK
  case 100000: // ERROR: Maximum value for short is 32767
}
```

Error 14024 **'case' outside switch statement.**

A `case` statement can appear only within a `switch` statement block.

Fix Make sure your braces are balanced correctly.

Error 14170 **catch not reached**

This `catch` clause will never be reached, since its exception is a subclass of an exception for a previous `catch` clause.

Fix Either reverse the order of the clauses or remove one.

Listing 4.43 Catching an exception that's already been caught

```
class MyException extends Exception { /* . . . */ }

public class CatchExample {
  public static void main(String args[]) {
    try {
      // . . .
    } catch (Exception e1) {
      // . .
    } catch (MyException e1) {
```

```
      // ERROR: This will never be reached, since it's a subclass
      //      of Exception, which is caught above
    }
  }
}
```

Error 14021 **"catch" without "try".**

A `catch` clause can appear only right after a `try` statement block.

Fix Make sure the is right after a `try` statement, and make sure your braces are balanced correctly.

Error 14089 ***className* must override *methodName* with a public method in order to implement *interfaceName*.**

In order to implement an interface, all methods of that interface must be declared public.

Listing 4.44 Failure to implement an interface by not declaring all methods of an interface public

```
  public interface feem {

  public void foo();
}

public class bar implements feem {

  void foo() {}  //must be specified public
}
```

Error 14017 **"class" or "interface" keyword expected.**

After `package` and `import` statements, a Java file can include only class and interface declarations.

Fix You cannot declare global variables in Java. Make sure your braces are balanced correctly.

Listing 4.45 Using unexpected keywords

```
package foo;
package bar; // ERROR: Extra package declaration.

int x; // ERROR: Variable declarations are not allowed here

public class A { /* . . . */ }// OK
```

Error 14260 **Class *className* not found.**

When the compiler fails to find a class, but the context of the request for the class is unknown, it generates this error.

Error 14117 **Class *symbol* already defined in *symbol*.**

You defined the class *class* more than one time in *symbol* .

Fix Remove or rename one of the class definitions.

Listing 4.46 Defining a class more than once

```
class RedundantClass { /* . . . */ }
class OKClass  { /* . . . */ }
class RedundantClass { /* . . . */ }// ERROR
```

Error 14167 **Class *symbol* can't be declared both abstract and final.**

You declared *class* to be both abstract and final. Such a class could never be used. An abstract class cannot be instantiated and must be subclassed. A final class cannot be subclassed.

Listing 4.47 Declaring a class to be both abstract and final

```
abstract final class Yuch { /* . . . */ } // ERROR
abstract class OK { /* . . . */ } // OK
final class AlsoOK extends OK { /*...*/ } // OK
```

Error 14191 **Class *symbol* not found in *location*.**

You used the class *class* in *location* without defining or importing it.

Fix Make sure you spelled the name correctly, defined it in the right place, or imported the class's package correctly.

Listing 4.48 Spelling a class name wrong

```
class Fubar { /* . . . */ }

public class StaticExample {
  public static void main(String args[]) {
    Foobar f = new Foobar();// ERROR: "Foobar" should be "Fubar"
  }
}
```

Error 14118 **Class name *class1* clashes with imported class *class2*.**

You defined the class *class2* that has the same name as the class *class1* that's in an imported package.

Fix Either rename or remove one of the classes, or don't import the package.

For example, say you have this class declaration in the package `foo`:

Listing 4.49 Declaring class B once

```
package foo;
public class B { /* . . . */ }
```

This class declaration would raise an error:

Listing 4.50 Declaring class B twice

```
package bar;
import foo.*;
```

```
class B { /* . . . */ } // ERROR: There's a class B in package foo
```

Error 14018 **Class or interface declaration expected.**

The compiler expected a class or interface declaration, but found something else instead. After a file's `package` and `import` statements, a file can contain only class and interface declarations.

Listing 4.51 Not finding a class or interface declaration

```
package bar;
import foo.*;

class Fubar { /* . . . */ }

int a;// ERROR: A variable definition is not allowed here.

public class StaticExample {
  public static void main(String args[]) {
       // . . .
  }
}
}     // ERROR: Extra closing brace
```

Error 14001 **Comment not terminated at end of input.**

You left out the final `*/` for a comment.

Listing 4.52 Forgetting to close off a comment.

```
public class A {
   // . . .
}

/*  Commenting out a class
class B {
   // . . .
} // ERROR: Forgot to close off the comment
```

Error 14112 **Constant expression required.**

In a `switch` statement block, one of the `case` labels is not a constant value.

Fix Either replace the `case` label with a constant, or change the `switch` statement to an `if-then-else` statement.

Listing 4.53 Using a variable or method call in a case label

```
public class TrivialApplication {
  final static int A = 2;

  public static void main(String args[]) {
    int b = 0, c = 1;

    switch (c) {
      case 100: // OK
      case A: // OK: Final variable is a constant
      case b: // ERROR: b is not a constant.
      case foo(A):  // ERROR: Method call is not a constant
    }
  }
}
```

Error 14160 **Constructor *constructor* requires a method body.**

You declared the constructor *constructor* without giving it a method body. All constructors must have method bodies, since you cannot declare a constructor to be either `abstract` or `native`.

Listing 4.54 Declaring a constructor without a method body

```
class A {
  int x;
  A();           // ERROR
  A() { x = 1; } // OK
}
```

Error 14066 **Constructor invocation must be the first thing in a method.**

If you explicitly call a constructor from another constructor, the explicit call must be the first statement.

Listing 4.55 Explicitly calling a constructor

```
class B extends A {
  int y;
  B() { y = 1; super(); }// ERROR
  B() { super(); y = 1; }// OK
}
```

Error 14085 **Constructor is ambiguous: *constructor1*, *constructor2***

There is more than one constructor that matches your constructor call. Neither constructor exactly matches the types of the call's arguments, but both are equally good matches after necessary conversions are applied to the call's arguments.

Fix Either explicitly cast one of the constructor call's arguments so it more closely matches one of the constructors, or (better yet) define a new constructor that more closely matches the constructor call's arguments.

Listing 4.56 Ambiguous constructors

```
class A {
  int x, y;
  A(short xx, int yy) { x = xx; y = yy; }
  A(int xx, short yy) { x = xx; y = yy; }
}

public class TrivialApplication {
  short m = 0, n = 1;
  A a1 = new A(m, n);
    // ERROR: No matter which constructor is used, one argument
    //      must be converted from a short to an int.
  A a2 = new A( (int)m, n );
```

```
    // OK: Second constructor now matches exactly
}
```

Error 14126 **Constructors can't be native, abstract, static, synchronized, or final: *constructor***

You declared a constructor to be `native`, `abstract`, `static`, `synchronized`, or `final`. You can declare constructors to be only `public`, `private`, or `protected`. Here are the reasons:

- `abstract` or `final`: You cannot override a constructor. Declaring it final is unnecessary. Declaring it abstract is useless, since you couldn't implement it in a subclass.
- `static`: You must invoke a constructor on a particular instance.
- `synchronized`: This is unnecessary since the object under construction is not available to other threads until the construction is complete.
- `native`: Allowing only Java constructors makes it easier for the Java Virtual Machine to ensure that superclass constructors are correctly invoked.

Error 14102 **'continue' must be in loop.**

A `continue` statement can appear only within a `while`, `do`, or `for` statement block.

Fix Either make sure your braces are balanced correctly, or rewrite your code to avoid the `break` statement.

Error 14233 **Couldn't find profiling classes; profiling data will not be generated.**

The option Generate Profiler Information was enabled in the Java Language settings panel, but the file `Profiler.zip` wasn't added to the project.

Error 14149 **Cyclic class inheritance.**

Your class inheritance includes a circular reference.

Listing 4.57 Cyclic class inheritance

```
class A extends B { /* . . . */ }
class B extends C { /* . . . */ }
class C extends A { /* . . . */ }
  // ERROR: Cyclic class inheritance
```

Error 14152 **Cyclic class inheritance or scoping.**

A general error is signaled when the compiler detects a cycle in an inheritance graph.

Error 14150 **Cyclic class inheritance: A subclass cannot enclose a superclass.**

Your class inheritance includes a circular reference when a subclass encloses a superclass.

Listing 4.58 Cyclic class inheritance when a subclass encloses a superclass

```
  public class foo extends feem {

    class feem {

    }
  }
```

Error 14151 **Cyclic interface inheritance.**

Your interface inheritance includes a circular reference.

Listing 4.59 Cyclic interface inheritance

```
interface A extends B { /* . . . */ }
interface B extends C { /* . . . */ }
interface C extends A { /* . . . */ }
  // ERROR: Cyclic interface inheritance
```

D to F (Java)

These are error messages that begin with *D*, *E*, or *F*.

Error 14025 **'default' outside switch statement.**

A `default` statement can appear only within a `switch` statement block.

Fix Make sure your braces are balanced correctly.

Error 14113 **Duplicate 'default' label.**

A `switch` statement contains more than one `default` label. Only one `default` label is allowed.

Listing 4.60 Using duplicate default labels

```
int a = 0;
switch (a) {
  case 1:    // OK
  default:   // OK
  case 2:    // OK
  default:   // ERROR
}
```

Error 14110 **Duplicate case label: *symbol***

You used the same `case` label twice in the same `switch` statement.

```
public class TrivialApplication {
  final static int B = 1;

  public static void main(String args[]) {
    int a = 0;

    switch (a) {
      case 1:  // OK
      case 2:  // OK
      case 1:  // ERROR
```

```
      case B:  // ERROR: B equals 1
    }
  }
}
```

Error 14142 **Duplicate inner class declaration: *innerClassName* is already defined in this scope.**

You cannot declare a duplicate of an inner class. An example of code which will generate this error is shown in Listing 4.61.

Listing 4.61 Illegally declaring a duplicate inner class

```
public class outerClass {

  class innerClass {

  }

  class innerClass {

  }
}
```

Error 14139 **Duplicate method declaration: *symbol***

A class has two methods with the same name and the same argument types.

Fix Either give one of the methods a different name, or use different types for the arguments.

Listing 4.62 Using duplicate method names and arguments

```
class A {
  void foo(int x)   { /* . . . */ }
  void foo(short y) { /* . . . */ }
    // OK: Java allows overloading:
    //   this has same name but different argument type
```

```
  void foo(int z)   { /* . . . */ }
    // ERROR: Same method name and same argument type
```

Error 14141 **Duplicate variable declaration: *symbol* was *symbol***

A class has two variables with the same name.

Fix Give one of the variables a new name.

Listing 4.63 Using duplicate variable names

```
class A {
  int a;
}

class B extends A {
  float a;  // OK: This shadows A.a
  float b;  // OK
  int b;      // ERROR
  native int b() ;// OK: But not a good idea. This is a method,
}                   // not a variable.
```

Error 14020 **"else" without "if".**

An `else` clause can appear only as part of an `if-then-else` statement block.

Fix Make sure your braces are balanced correctly. If the then clause contains more than one statement, make sure they're enclosed in braces.

Listing 4.64 Using an else clause without an if statement

```
if (a==0)
  System.out.println ("Hello");
  System.out.println ("How are you?");
else
```

```
  // ERROR: The then clause is not enclosed in brackets.
  System.out.println ("Goodbye");
```

Error 14198 **Exception *symbol* can't be thrown in initializer.**

While initializing a variable, you used a method that throws an exception that isn't caught.

Fix Either rewrite the method to make sure that exception is caught, or initialize the variable differently.

Listing 4.65 Throwing an exception in an initializer

```
class E1 extends Exception { /* . . . */ }
class E2 extends Exception { /* . . . */ }

class A {
  int a = foo(1);// ERROR: foo() throws an exception that
                         //     is not caught.
  int foo(int x) throws E2 {
    try {
      if (x==0)
        throw new E1();// OK: E1 is caught
      else if (x<0)
        throw new E2();// ERROR: E2 is NOT caught
    } catch (E1 e) {
      System.out.println("oops");
    } finally {
      return x;
    }
  }
  // . .
}
```

Error 14205 **Exception *exception* is never thrown in the body of the corresponding try statement.**

In a `catch` clause, you try to catch an exception that is never thrown.

Fix You may have forgotten to remove this `catch` clause after removing the code that throws that exception, or you may have gotten the name of the exception wrong.

Listing 4.66 Catching an exception that is never thrown

```
class E1 extends Exception { /* . . . */ }
class E2 extends Exception { /* . . . */ }

class A {
  int foo(int x) {
    try {
      if (x==0)
        throw new E1();
    } catch (E2 e) {// ERROR: E2 is not thrown.
      System.out.println("oops");
    } finally {
      return x;
    }
  }
}
```

Error 14262 **Exception *exceptionType* must be caught, or it must be declared in the throws clause of this constructor.**

If a constructor throws a checked exception, it must declare it.

For example:

```
class A {
 A() { //need throws clause here.
   throw new java.io.IOException();
 }
}
```

Error 14204 **Exception *symbol* must be caught, or it must be declared in the throws clause of this method.**

You throw an exception that is never caught and is not declared in the method's `throws` clause.

Fix Either put the code in a `try` statement that catches the exception, or add the exception to the method's `throw` clause.

Listing 4.67 Throwing an uncaught, undeclared exception

```
class E1 extends Exception { /* . . . */ }

class A {
  int foo(int x) {
    if (x==0)
      throw new E1(); // ERROR: Not caught and
    else          //     not in throws clause
      return x;
  }
}
```

Error 14022 **"finally" without "try".**

A `finally` clause can appear only after a `try` or `catch` statement block.

Fix Make sure the clause follows a `try` or `catch` statement, and make sure your braces are balanced correctly.

Error 14189 **File *fileName* does not contain *className* as expected, but *badClassName*.**

This will occur if a zip contains an entry for a class of a given name, but upon loading the class, the compiler determines that it has a different name.

Fix Either remove the `final` modifier or change the name of one of the classes.

Listing 4.68 Overriding a final method

```
class A {
  final void x() { /*...*/ }
  void y() { /*...*/ }
}
```

```
class B extends A {
  void x() { /*...*/ } // ERROR: x() is final in A
  void y() { /*...*/ } // OK: y() is not final
}
```

Error 14256 **Floating point arithmetic may not be both strict and wide.**

The modifiers `strictfp` and `widefp` may not be both applied to a method. For example:

```
class A {
  strictfp widefp float method() { } //error
}
```

G to I (Java)

These are error messages that begin with *G*, *H*, or *I*.

Error 14213 **Hexidecimal numbers must contain at least one hexidecimal digit.**

This error is given when your code contains a hexadecimal literal with no numbers, such as

```
int i = 0x;
```

Error 14026 **I/O error in *symbol.***

If the compiler has trouble reading from a file (`.java`, `.class`, `.jar`, or `.zip`), it might report this error. This could mean

- the file is damaged,
- the file couldn't be found, or
- the compiler expected the file to be something it was not.

This error should be rare in occurence. Since you've found an instance where it was reported to you, please send a bug report to Metrowerks technical support.

Error 14016 **Identifier expected.**

The compiler could not find an identifier where it expected one.

Fix Maybe you forgot to put in an identifier where one is needed, or you accidentally used a reserved word to name a variable, method, class, or interface.

Listing 4.69 Abusing a reserved word

```
class A {
  int ;                    // ERROR: Forgot to specify names.
  void float() { /* . . . */ }// ERROR: float is a reserved word
}
```

Error 14036 **Impossible for *symbol* to be instance of *symbol*.**

If the compiler can determine that an instance of test will always fail, it will issue this error.

Listing 4.70 Impossible for *symbol* to be instance of *symbol*.

```
String i;

if (i instanceof java.awt.Component) //can never be true
  return;
```

Error 14087 **Incompatible type for *location*. Can't convert *type1* to *type2*.**

You are trying to convert a variable of *type1* to be *type2*, but the types are incompatible. You cannot perform the conversion even with an explicit conversion. The *location* is where the conversion is taking place: in a declaration, = statement, etc.

Listing 4.71 Incompatible conversions

```
class A { /* . . . */ }
class C { /* . . . */ }
```

```
public class TrivialApplication {
  public static void main(String args[]) {
    C cc;
    A a = new A(),
    A aa = new C();// ERROR
    cc = a;   // ERROR
  }
}
```

Error 14086 **Incompatible type for *location*. Explicit cast needed to convert *type1* to *type2*.**

You are trying to convert implicitly a variable of *type1* to be *type2*, but you need to perform an explicit conversion. The *location* is where the conversion is taking place: in a declaration, = statement, etc.

Listing 4.72 Incompatible implicit conversions

```
class A { /* . . . */ }
class B extends A { /* . . . */ }

public class TrivialApplication {
  public static void main(String args[]) {
    A a = new A();
    B bb;
    short x;

    bb = a;       // ERROR: The conversion is OK,
                        //    but it must be explicit
    bb = (B) a;// OK: Now it's explicit
    x = Math.PI; // ERROR
    x = (short) Math.PI; // OK
  }
}
```

Error 14146 **Inconsistent member declaration. At most one of public, private, or protected may be specified.**

The compiler generates an error when an inconsistent member declaration is found.

Error 14220 **Initializer must be able to complete normally.**

This error occured because static and instance initializers must be able to complete without throwing an exception.

For example:

```
public class foo {

{
  throw new Error();
}

}
```

would result in a compile time error.

Error 14135 **Inner classes can't be volatile, transient, native, or synchronized: *innerClassName***

An error occurs when an inner class is volatile, transient, native or synchronized.

Error 14042 **Inner type *innerClassName* in *className* not accessible from *className*.**

Code in one class attempted to reference an inaccesible inner class of another class.

Listing 4.73 Attempting to reference an inaccessible inner class of another class

```
public class bar {

  private static class foo {

    public static void aMethod() { }
  }

}

public class blat {

  public void fromHere() {

    bar.foo.aMethod(); //foo is not accesible from blat
  }
}
```

Error 14121 **Instance methods can't be overridden by a static method. Method *method* is an instance method in *class*.**

You tried to override the instance method *method* in class *class* with a static method. Only instance methods may override instance methods.

Fix Change either the name or the declaration of one of the methods.

Listing 4.74 Overriding an instance method with a static method

```
class A {
  void x() { /* . . . */ }
  void y() { /* . . . */ }
 }

class B extends A {
  static void x() { /*...*/ } // ERROR: x() is an instance
                              //      method in A
```

```
  void y() { /*...*/ }// OK
}
```

Error 14161 **Instance variables can't be void: *variable***

You declared the instance variable *variable* to be void. Only methods may be void.

```
class A {
  void x;                   // ERROR
  void x() { /* . . . */ } // OK
}
```

Error 14237 **Integer literal out of range. Decimal int literals must be in the range -2147483648 to 2147483647.**

You have erroneously declared a decimal integer literal to have a range greater than the range allowed by the compiler.

Error 14240 **Integer literal out of range. Decimal long literals must be in the range -92233720368547758O8L to 92233720368547758O7L.**

You have erroneously declared a decimal long literal to have a range greater than the range allowed by the compiler.

Error 14238 **Integer literal out of range. Hexadecimal int literals must be in the range 0x0 to 0xffffffff.**

You have erroneously declared a hexadecimal integer literal to have a range greater than the range allowed by the compiler.

Error 14241 **Integer literal out of range. Hexadecimal long literals must be in the range 0x0L to 0xffffffffffffffffL.**

You have erroneously declared a hexadecimal long literal to have a range greater than the range allowed by the compiler.

Error 14236 **Integer literal out of range. Octal int literals must be in the range 00 to 037777777777.**

You have erroneously declared an octal integer literal to have a range greater than the range allowed by the compiler.

Error 14239 **Integer literal out of range. Octal long literals must be in the range 00L to 01777777777777777777777L.**

You have erroneously declared an octal long literal to have a range greater than the range allowed by the compiler.

Error 14166 **Interface *interface* of *location* not found.**

You used the class *interface* in *location* without defining or importing it.

Fix Make sure you spelled the name correctly, defined it in the right place, or imported the interface's package correctly.

Error 14157 **Interface *symbol* repeated.**

In the implements clause of an class definition, you used the same interface more than once.

Fix Remove one of the occurrences.

Listing 4.75 Listing the same interface twice

```
interface CanFly  { /* . . . */ }
interface CanWalk { /* . . . */ }

class Bird implements CanWalk, CanFly, CanWalk { /* . . . */ }
  // ERROR: CanWalk is repeated twice
```

Error 14129 **Interface fields can't be private or protected: *symbol***

In an interface declaration, you declared a field to be private or protected. All interface fields are public. You can explicitly declare a field with the modifier `public`, but it is unnecessary and considered to be bad programming style.

Listing 4.76 Declaring an interface field to be private

```
interface A {
  private int CONST1 = 1;// ERROR
  public  int CONST2 = 2;// OK, but unnecessary
  int CONST3 = 3;// BETTER
 }
```

Error 14128 **Interface methods can't be native, static, synchronized, final, private, or protected : *symbol***

You declared an interface method to be native, static, synchronized, final, private, or protected. All interface methods are public and static. You can explicitly use the modifiers `public` and `abstract`, but it is unnecessary and considered to be bad programming style.

Here is why you cannot use the specified modifiers:

- `private` and `protected`: All instance methods are public.
- `static`: All instance methods are abstract.
- `final`: All instance methods are abstract. However, when you define a class that implements the interface, you can implement an interface method with a final method.
- `native` and `synchronized`: These describe how the method is implemented, which an interface does not specify. However, when you define a class that implements the interface, you can implement an interface method with a native or synchronized method.

Listing 4.77 Declaring an interface method to be final

```
interface A {
  final  int meth1();  // ERROR
  abstract int meth2();// OK, but unnecessary
  int meth3();                // BETTER
 }

class B implements A {
  public final int meth2() { return 0; } // OK
```

```
  // . . .
}
```

Error 14154 **Interfaces can't be final: *symbol***

You cannot declare an interface to be final. All interfaces are abstract. You can explicitly use the modifiers `abstract`, but it is unnecessary and considered to be bad programming style.

Listing 4.78 Declaring an interface to be final

```
final interface A { }  // ERROR
abstract interface B { } // OK, but unnecessary
interface C { }  // BETTER
```

Error 14125 **Interfaces can't have constructors.**

An interface cannot contain constructors. It can contain only method and field declarations.

Error 14127 **Interfaces can't have static initializers.**

An interface cannot contain static initializers. It can contain only method and field declarations.

Error 14063 **Invalid argument type *symbol* for *symbol*.**

If an operator is passed a bad type, this error will be signaled.

```
String s = "hi";
s++; //error
```

Error 14034 **Invalid arguments to *symbol*.**

If an operator is given arguments of a type it cannot handle, this error will be signaled.

For example:

```
int i = "hi" << 3;  //can only shift ints or longs
```

Error 14084 **Invalid array dimension.**

If an array declaration has an initializer expression, its dimensions should be left unspecified.

```
int i[] = {1,2,3};     //ok
int i[] = new int[3];  //ok

int i[3] = {1,2,3};    //error
int i[3] = new int[3]; //error
```

Error 14035 **Invalid cast from *type1* to *type2*.**

You tried to make an illegal conversion from *type1* to *type2*. This conversion isn't allowed even with an explicit cast.

```
class A { /* . . . */ }
class B { /* . . . */ }

public class TrivialExample {
  public static void main(String args[]) {
    B b = new B();
    A a = (A)b;// ERROR: Can't cast one unrelated
                        //     class to another
    int c = (int)"123";// ERROR: Can't cast a string to an int
  }
}
```

Error 14004 **Invalid character constant.**

You created a character constant incorrectly. A character constant must be a single character or a single escape sequence between two single quotes.

Listing 4.79 **Creating invalid character constants**

```
int a = 'TEXT'; // ERROR: Can only be one char
char b = 'T'; // OK
char c = '\uFFFFF'; // ERROR: This is two chars (\uFFFF and F)
```

```
char d = '\uFFFF';// OK
char e = 'a"; // ERROR: Ends in double quote
```

Error 14009 **Invalid character in input.**

You used an invalid character in your code.

Fix Perhaps you intended to use it in a comment, string, or character constant.

Listing 4.80 Using an invalid character

```
public class TrivialExample {
  public static void main(String args[]) {
    char c = '\u000E'; // OK
    \u000E             // ERROR: Can't use this character alone in
  }                    //        a file
}
```

Error 14008 **Invalid character in number.**

You used an invalid character in a numeric constant. A decimal number may contain only the digits `0`–`9` and cannot begin with `0`. An octal number begins with `0` and may contain only the digits `0`–`7`. A hexadecimal number begins with `0x` or `0X` and may contain the digits `0`–`9`, the letters `a`–`f` and the letters `A`–`F`.

Listing 4.81 Using invalid characters in numbers

```
int a = 0x1F2G; // ERROR: G is not a hex digit
int b = 1OOO; // ERROR: You used letter O, instead of zero (0)
```

Error 14007 **Invalid character in octal number.**

You used an invalid character in an octal integer constant. An octal number begins with `0` and may contain only the digits `0`–`7`. A decimal number must not begin with 0, since the compiler will assume it's an octal constant.

Listing 4.82 Using invalid characters in octal numbers

```
int a = 079; // ERROR: 9 is not an octal digit.
int b = 077; // OK
int c = 79;  // OK
```

Error 14158 **Invalid class file format: *symbol*, *symbol***

If the compiler fails to load a binary class from a `.class`, `.jar`, or `.zip` file, this error is issued with a description of what is wrong with the class in question.

Error 14104 **Invalid declaration.**

If a variable is declared in an illegal place, this error is issued.

An example of this error is shown in Listing 4.83.

Listing 4.83 Invalid declaration

```
  try {

  }
  catch( Exception e, int i) { //can only have one declaration in
                               //catch stat

  }
```

Error 14006 **Invalid escape character.**

You used an invalid escape character. The valid escape sequences are `\b`, `\t`, `\n`, `\f`, `\r`, `\"`, `\'`, `\\`, and Unicode escape sequences. A valid Unicode escape sequence is `\u` followed by four hexadecimal digits.

Listing 4.84 Using an invalid escape sequence

```
String s = "\i"; // ERROR
char c1 = '\u000G';// ERROR: G is not a hex digit
char c2 = '\u000A';  // OK
```

Error 14169 **Invalid expression statement.**

There's a statement that contains only a single expression which isn't allowed to be used alone in a statement. The only expressions you can use alone in a statement are assignment, method call, class instance creation, pre-increment, pre-decrement, post-increment, and post-decrement.

Listing 4.85 Using invalid expression statements

```
public class TrivialExample {
  static int twice(int y) { return 2*y; }

  public static void main(String args[]) {
    int x = 0;

    (void) twice(3); // ERROR: (void) isn't allowed
    twice(3);  // OK
    x;              // ERROR: A variable alone is not a statement
    x++;          // OK: Post-increment is OK
  }
}
```

Error 14010 **Invalid floating point format.**

You used an invalid character in a floating-point constant.

Listing 4.86 Invalid floating point format

```
float f = 1.0e--30;// ERROR
```

Error 14032 **Invalid initializer for type *symbol*.**

You used an array initializer to initialize something that isn't an array.

Listing 4.87 Using an array initializer incorrectly

```
int a = { 1, 2, 3 };  // ERROR
int[] b = { 1, 2, 3, {1, 2, 3} };// ERROR
int[][] c = { { 1, 2, 3}, { 1, 2, 3 } }; // OK
int[] d = { 1, 2, 3 }; // OK
```

Error 14108 **Invalid label.**

You named a label incorrectly. Labels use the same naming conventions as other identifiers.

Listing 4.88 Using an invalid label

```
for (int i =1; i<10; i++ ) {
  1a: i--; // ERROR
  a1: i--; // OK
  // . . .
}
```

Error 14033 **Invalid left hand side of assignment.**

Only local variable, field, or array access may appear in the left hand side of an assignment.

Fix Perhaps you intended to use == instead of =.

Listing 4.89 Using an invalid left hand side of an assignment

```
public class TrivialExample {
  static int twice(int x) { return 2*x; }

  public static void main(String args[]) {
    if (twice(3) = 6)  /* . . . */ ;// ERROR
```

```
    if (twice(3) == 6) /* . . . */ ;// OK
  }
}
```

Error 14163 **Invalid method declaration; method name required.**

When you declared a method, you left out the method name.

Listing 4.90 Method has no name

```
public class A {

  public void (int x) { //method has no name

  }
}
```

Error 14162 **Invalid method declaration; return type required.**

When you declared a method, you left out the return type. If the method returns no value, you must declare it to be void.

Listing 4.91 Leaving out the return type

```
hello(int x) { System.out.println("Hello"); } // ERROR
void bye(int x) { System.out.println("Bye"); }// OK
```

Error 14164 **Invalid qualified constructor name.**

Your qualified constructor name is invalid. In Listing 4.92, the constructor illegally has a return type.

Listing 4.92 Constructor has a return type

```
  public class A {

    public void A() { //constructor can't have return type
```

```
    }
  }
```

Error 14088 **Invalid term.**

If a type expression is found where a value should be, this error is signaled. An example of this error is found in Listing 4.93

Listing 4.93 Invalid term.

```
int i = int;
```

Error 14037 **Invalid type expression.**

If the compiler expects to find a type, but instead finds an expression that cannot represent a type, it will issue this error.

Listing 4.94 Invalid type expression

```
  String a = "hi";

  Class aClass = a.class;  //Should use "String.class"
```

J to N (Java)

These are error messages that begin with *J, K, L, M,* or *N*.

Error 14179 **Local class *innerClassName* is already defined in this method.**

You declared a local class twice in the same method.

Listing 4.95 Declaring a local class twice in the same method

```
public class foo {

  public void aMethod() {

    class x extends y {
```

```
    }

    class x extends z {
    }
  }
}
```

Error 14221 **Member interfaces can only occur in interfaces and top-level classes.**

Inner classes (nested classes that are not static) cannot contain interfaces. Examples of legal and illegal member interfaces are shown below.

```
public class outerClass {
  public class innerClass {
    interface anInterface { } //illegal
  }

  interface anotherInterface { } //OK, in top level class

  public static class nestedTopLevel {

    interface yetAontherInterface { } //also OK
  }
}
```

Error 14212 **Method *methodName* can't be static in *innerClassName*. Only members of interfaces and top-level classes can be static.**

Non-top-level classes can't contain static variables.

```
public class outerClass {

  public class innerClass {

    static int foo() { }
      //illegal; innerClass isn't top level
      //(i.e. isn't package level and isn't static)
```

```
  }
}
```

Error 14130 **Method *method* can't be transient. Only variables can be transient.**

A method cannot be declared transient. A transient variable is one that isn't saved permanently (to a file, for example) when the rest of the object's variables are saved. Methods are not saved.

Error 14131 **Method *method* can't be volatile. Only variables can be volatile.**

A method cannot be declared volatile. A volatile variable is one whose value may change unexpectedly. A method doesn't have a value.

Error 14183 **Method *methodName* is inherited in *className*, and hides a method of the same name in *anscestorClassName*. An explicit "this" qualifier must be used to select the desired instance.**

An example of this error and its fix are shown in Listing 4.96.

Listing 4.96 Failure to invoke a method

```
public class bar {

  public int foo() { }

}

public class feem {

  int foo() { }

  public void aMethod() {

    bar aBarSubclass = new bar() {

      public void innerMethod() {
```

```
        int x = foo(); //if bar.foo() is meant to be invoked,
                       //bar.this.foo() should be used
      }
    }
  }
}
```

Error 14069 **Method *symbol* in *symbol* is not accessible from *symbol*.**

If a method cannot be invoked because of it's access, this error is signaled.

```
class A { private void foo(); }

class B { { foo(); }} //error
```

Error 14068 **Method *symbol* not found in *symbol*.**

You refer to a method that is not defined in the current class. You may have spelled the method's name wrong, forgot to define the method, or meant to refer to a method in another class.

Listing 4.97 Using a misspelled method

```
class A {
  void bar() { /* . . . */ }
  protected void foo() {
    baz();// ERROR: Maybe you meant bar()
  }
}
```

Error 14159 **Method *symbol* requires a method body. Otherwise declare it as abstract.**

If a method is not declared to be abstract or native, you must define it with a method body.

Fix Either declare it to be abstract or native, or define the method.

Listing 4.98 Failing to define a method

```
class A {
  void foo1();     // ERROR
  void foo2() { /* . . . */ } // OK
  abstract void foo3(); // OK
  native void foo4(); // OK
}
```

Error 14119 **Method redefined with different return type: *method1* was *method2***

You tried to override*method2* with *method1,* which has the same name and argument types as *method2* but a different return type. When you override one method with another, their return types must match. If you want to overload one method with another, their argument types must be different.

Listing 4.99 Overloading a method with a different return type

```
class A {
  int y;
  int foo(int x) { return x*y; }
}

class B extends A {
  float w;
  float foo(int x) { return x*w; } // ERROR: Same argument types,
                                   //     different return type
  int foo(int x) { return x+w; } // OK: This overrides foo()
  float foo(float x) {return x*w;} // OK: This overloads foo()
}
```

Error 14124 **Methods can't be overridden to be more private. Method *method* is not private in *class*.**

You tried to override method with another method that is more private. The overriding method must be just as private or less private than the original method.

Listing 4.100 Overriding a method with a more private method

```
class A {
  void meth1() { /* . . . */ }
  void meth2() { /* . . . */ }
  void meth3() { /* . . . */ }
  void meth4() { /* . . . */ }
}

class B extends A {
  private void meth1() { /* . . . */ } // ERROR
  void meth2() { /* . . . */ } // OK
  protected void meth3() { /* . . . */ } // OK
  public void meth4() { /* . . . */ }// OK
}
```

Error 14123 **Methods can't be overridden to be more private. Method *symbol* is protected in *symbol*.**

You tried to override a protected method with another method that is private. The overriding method must be just as private or less private than the original method.

Listing 4.101 Overriding a protected method with a private method

```
class A {
  protected void meth1() { /* . . . */ }
  protected void meth2() { /* . . . */ }
  protected void meth3() { /* . . . */ }
  protected void meth4() { /* . . . */ }
}

class B extends A {
  private void meth1() { /* . . . */ } // ERROR
  void meth2() { /* . . . */ } // ERROR
  protected void meth3() { /* . . . */ } // OK
  public void meth4() { /* . . . */ }// OK
}
```

Error 14122 **Methods can't be overridden to be more private. Method *symbol* is public in *symbol*.**

You tried to override a public method with another method that is protected or private. The overriding method must be just as private or less private than the original method.

Listing 4.102 Overriding a protected method with a private method

```
class A {
  public void meth1() { /* . . . */ }
  public void meth2() { /* . . . */ }
  public void meth3() { /* . . . */ }
  public void meth4() { /* . . . */ }
}

class B extends A {
  private void meth1() { /* . . . */ } // ERROR
  void meth2() { /* . . . */ } // ERROR
  protected void meth3() { /* . . . */ } // ERROR
  public void meth4() { /* . . . */ }// OK
}
```

Error 14140 **Methods can't be redefined with a different return type: *method1* was *method2***

You tried to overload *method2* with *method1*, which has the same name and argument types as *method2* but a different return type. When you overload one method with another, their argument types must be different.

Listing 4.103 Overloading a method with a different return type

```
class A {
  int y;
  float w;
  int foo(int x) { return x*y; }

  float foo(int x) { return x*w; } // ERROR: Same argument types,
                                   //     different return type
```

```
  float foo(float x) {return x*w;} // OK: This overloads foo()
}
```

Error 14019 **Missing term.**

The compiler didn't encounter a term (such as an identifier) where it expected one.

Fix Check for a typing error.

Listing 4.104 Missing terms

```
int a === 1, b, c;// ERROR: Too many equal signs
foo(a,,c);// ERROR: Too many commas
```

Error 14156 **Multiple inheritance is not supported.**

Java does not allow multiple inheritance: a class extending more than one other class.

Fix Rewrite your code using interfaces instead.

Listing 4.105 Using multiple inheritance

```
class A { /* . . . */ }
class B { /* . . . */ }
class C extends A, B { /* . . . */ }// ERROR

interface X { /* . . . */ }
interface Y { /* . . . */ }
class Z implements X, Y { /* . . . */ } // OK
```

Error 14095 **No constructor matching *constructor* found in *class*.**

The compiler couldn't find a constructor that matches the arguments you used in a new instance creation expression (such as in a `new` statement).

Fix Either cast the arguments to match a constructor or create a new constructor for the argument types.

Listing 4.106 Finding no constructor that matches call

```
class A {
  int x;
  A(int xx) { x = xx; }
  // . . .
}

public class TrivialApplication {
  public static void main(String args[]) {
    A a1 = new A();// ERROR: Since you defined a construc-
                          //     tor, compiler doesn't create
                          //     default constructor.
    A a3 = new A(2.0); // ERROR: No constructor for float arg.
    A a4 = new A((int)2.0);// OK: Uses constructor for int arg.
    A a2 = new A(2); // OK
  }
}
```

Error 14207 **No enclosing instance of *className* is in scope; an explicit one must be provided when creating *innerClassName*, as in `outer. new Inner()` or `outer. super()`.**

If an inner class is created without an implicit instance of the outer class (i.e. is not created from within an instance method of the outer class), the new (or super) keyword must be qualified.

Listing 4.107 No enclosing instance of the class is in scope

```
  public class foo {

    public class bar {
    }

    public static bar makeBar(foo outerInstance) {

      return new bar(); //illegal, as there is no implicit
```

```
                        //instance of foo.  Instead, use
                        //"return foo.new bar();"
    }
  }
```

Error 14209 **No enclosing instance of *className* is in scope; an explicit one must be provided when accessing *memberName*, as in `outer.member`.**

A static inner class is attempting to reference an instance variable of the outer class without qualifing the enclosing instance.

Listing 4.108 No enclosing instance of a class is in scope

```
public class foo {

  int bar; //instance variable

  static class blat {

    public void aMethod(foo aFoo) {

      bar++; //no enclosing instance of foo.
             //Instead, use "aFoo.bar++;"
    }
  }
}
```

Error 14208 **No enclosing instance of *className* is in scope; cannot create a default constructor for *className*.**

In the constructor for a subclass of an inner class, if there is no enclosing instance of the outer class in scope, and the outer class has no public paramater-less constructor, this error will be generated.

Listing 4.109 Cannot create a default destructor for the class

```
public class foo {
```

```
  public foo(String s) { //Note that foo requires a string to be
                         //constructed
  }

  public class bar {
  }
}

public class blat extends foo.bar {
}

public class feem {

  public static blat createABlat() {
    return new blat(); //no enclosing instance of foo, and bar
       //needs one, and one can't be synthesized by the
       //compiler because foo has no default constructor
  }
}
```

Error 14100 **No label definition found for *label*.**

You refer to the label *label* but do not define it.

Fix Make sure the label name is spelled correctly.

Listing 4.110 Finding no matching label

```
int i = 0;
while (true) {
  System.out.println(" i = " + i );
  i++;
  if (i>10) break end; // ERROR: No label named end.
}
```

Error 14094 **No method matching *method* found in *class*.**

You tried to access the *method* in *class,* but the compiler cannot find any method with those argument types in that class.

Fix Make sure you spelled the name of the class correctly, used the correct arguments, and are using a method you have access to.

Listing 4.111 Finding no matching method

```
class B {
  private void foo() { /* . . . */ }
  void bar() { /* . . . */ }
}

public class TrivialApplication {
  public static void main(String args[]) {
    B b = new B();

    b.foo();  // ERROR: bar() is private
    b.bar(3); // ERROR: foo() doesn't have any arguments.
    b.bar();  // OK
  }
}
```

Error 14040 **No variable *symbol* defined in *symbol*.**

You tried to access the *variable* in *class*, but the compiler cannot find any variable with that name in that class.

Fix Make sure you spelled the name of the class correctly.

Listing 4.112 Finding no matching variable

```
class B {
  int x;
  // . . .
}

public class TrivialApplication {
  public static void main(String args[]) {
    B b = new B();

    int y = b.xx; // ERROR: xx is misspelled
    int z = b.x;  // OK
```

```
  }
}
```

Error 14076 **Note: {0} has been deprecated.**

This warning means that the item in question has been deprecated. You can turn this warning off by disabling the option Emit deprecation warnings in the Java Language settings panel.

Error 14078 **Note: The constructor {0} has been deprecated.**

This warning means that the item in question has been deprecated. You can turn this warning off by disabling the option Emit deprecation warnings in the Java Language settings panel.

Error 14077 **Note: The method {0} in {1} has been deprecated.**

This warning means that the item in question has been deprecated. You can turn this warning off by disabling the option Emit deprecation warnings in the Java Language settings panel.

Error 14079 **Note: The variable {0} in {1} has been deprecated.**

This warning means that the item in question has been deprecated. You can turn this warning off by disabling the option Emit deprecation warnings in the Java Language settings panel.

Error 14214 **Note: Method *methodName* in *className* does not override the corresponding method in *className*, which is private to a different package.**

This lets you know that you may think you are overriding a method of a superclass, but that the superclass method in question is not overridable from your package, and you are thus really introducing a new method of the same name and signature.

Listing 4.113 Method fails to override the corresponding method

```
package A;

public class blat {
```

```
  void foo() {}
}

//new file

package B;

public class bar {

  void foo() {};
  //foo is not overridable, as it is package-private.
}
```

Error 14206 **Note: The cloning of an array does not throw any checked exceptions, and therefore does not require any catch clauses. Please remove unused catch clauses, or if you wish to retain compatibility with older compilers, you may insert an artificial throw as follows: if (false) throw new CloneNotSupportedException();**

In 1.1, clone()-ing an array no longer throws a checked exception. However, 1.0.2 code requries a clone() invocation to be surrounded with a try block, as shown in Listing 4.114.

Listing 4.114 Using an artificial throw to retain compatibility with older compilers

```
try {

  int[] foo = {1,2,3}

  int[] fooCopy = foo.clone();
}
catch (CloneNotSupportedException e) {
}
```

Error 14243 **Numeric overflow: double literal too large.**

A double literal value was specified that is too large to be represented by the given type.

Error 14245 **Numeric underflow: double literal too small.**

A double literal value was specified that is too small to be represented by the given type.

Error 14242 **Numeric overflow: float literal too large.**

A float literal value was specified that is too large to be represented by the given type.

Error 14244 **Numeric underflow: float literal too small.**

A float literal value was specified that is too small to be represented by the given type.

O to R (Java)

These are error messages that begin with *O*, *P*, *Q*, or *R*.

Error 14065 **Only constructors can invoke constructors.**

You used a constructor call (`this()` or `super()`) in a method, which is not a constructor.

Fix Consider removing the constructor call with a `new` statement.

Listing 4.115 Calling a constructor from a method

```
class A {
  int x;
  A(int xx) { x = xx; }
  A() { this(0); }      // OK: A() is a constructor
  A foo(int xx) { return this(0); }// ERROR: foo() is a method
  A bar(int xx) { return new A(xx); }// OK: bar() uses new.
}
```

Error 14038 **Only named classes can have "extends" or "implements" clauses.**

Anonymous block local classes, which are created in `new` expressions, cannot use the `extends` or `implements` keyword in order to specify their origin. Instead, the superclass or interface must be declared in the new instance expression.

For instance, an anonymous class which is to implement the runnable interface should appear as:

```
Runnable r = new Runnable() {  /*class body*/ }
```

rather than

```
Runnable r = new Object extends Runnable { /* class body */ }
```

Error 14116 **Only one package declaration allowed.**

If a file has two or more package declarations, this error is signaled.

Error 14261 **Package java.lang not found. Please make sure classes.zip is added to the project.**

This error is issued when the compiler cannot find the basic classes in `java.lang`.

Error 14192 **Package *symbol* not found in *symbol*.**

If an import statement occurs for a non-existent package, this error will occur.

Error 14174 **Public *symbol* must be defined in a file called "*symbol*".**

This warning will only be issued when the option Strict Filenames is enabled in the **Java Language** settings Panel. It occurs if a public class does not reside in a file of the same name.

Error 14188 **Recursive constructor invocation: *symbol*.**

You have a constructor that calls itself, which would cause infinite conversion.

Listing 4.116 Defining a recursive constructor

```
class A {
  int x;
  A(int xx) { this(xx); }// ERROR: Recursive!
  A(byte xx) { this( (short) xx ); } // OK: Because of cast, it
                                     //  calls the constructor
                                     //  below
  A(short xx) { x = xx; }
}
```

Error 14075 **Reference to method *methodName* in *className* as if it were a variable.**

The compiler generates an error when you reference a method in a certain classname as if it were a variable, as shown in Listing 4.117.

Listing 4.117 Referencing a method in a classname as if it were a variable

```
public class foo {

  public int feem() {
    return 0;
  }


  public int bar() {

    int x = feem;  //should be "int x = feem();"
  }
}
```

Error 14045 **Reference to *symbol* is ambiguous. It is defined in *symbol* and *symbol*.**

If a class implements two interfaces, each of which contain a field of a given name, any references to that variable must be qualified with the interface name.

Listing 4.118 Reference to *symbol* is ambiguous. It is defined in *symbol* and *symbol*.

```
interface A { String name; }

interface B { String name; }

class C implements A,B {

  String aName = name; // error here, should be "A.name" or
"B.name"
```

```
}
```

Error 14074 **Reference to variable *symbol* in *symbol* as if it was a method.**

If a name resolves to a variable, but it is used as a method, this error is signaled.

```
class A {

  Object foo;

  public Object method() {
    return foo(); //error here
  }
}
```

Error 14145 **Repeated modifier.**

You repeated the same modifier in a declaration. You can use a modifier only once.

Listing 4.119 Repeating a modifier

```
public public class A { // ERROR
  final public final int X = 1; // ERROR
}
```

Error 14064 **"length" applied to *symbol*, which is not an array.**

If an expression attempts to reference the length of an object that is not known to be an array, this error is signaled.

```
Object foo = new int[i];

int i = foo.length; //Static type of foo is Object, although
                    //dynamic type is int[]
```

Error 14107 **'return' inside static initializer.**

A static initializer cannot return a value, so it cannot contain a `return` statement.

Listing 4.120 No return statement allowed in static initializer

```
public class X{
  // . . .
  static {
    System.out.println ("Class X is loading..");
    return;
    // ERROR: Static initializer cannot contain a return statement
  }
}
```

Error 14109 **Return required at end of *method*.**

You declared that a method returns a value, but it doesn't have a `return` statement at each place where the method might exit.

Listing 4.121 Missing a return statement

```
int foo() { /*...*/ } // ERROR: No return statement

int bar(int x) {
  if (x<=0)
    return 0;
}                    // ERROR: No return statement for x > 0.
```

Error 14105 **'return' with value from *method*.**

You declared the method *method* to be `void`, but it contains a `return` statement with a value.

Fix Either remove the value from the `return` statement, or declare the method differently.

Listing 4.122 Returning a value from a void method

```
void twice (int a) {
  return (a*2);
  // ERROR: Either declare twice to return an int
  //      or remove the value from the return statement
}
```

Error 14115 **'return' with value from constructor: *constructor***

You cannot return a value from a constructor.

Listing 4.123 Returning a value from a constructor

```
public class Bird {
  // . . .
  int Age;
  Bird (int x) {
    this.Age = x;
    return this; // ERROR
  }
}
```

Error 14106 **'return' without value from *symbol*.**

You declared the method *method* to return a value, but it contains a `return` statement with no value. Either add a value to the `return` statement, or declare the method to be `void`.

Listing 4.124 Returning no value from a function

```
int factorial(int x) {
  if (x==1) return;  // ERROR
  else return (x * factorial(x-1));// OK
}
```

S to U (Java)

These are error messages that begin with *S*, *T*, or *U*.

Error 14014 **Statement expected.**

The compiler expected a statement, but didn't see one.

Listing 4.125 Not finding a statement where expected.

```
class A {
  void foo(int x) {
      class B { } // ERROR: Not a statement
  }
}
```

Error 14171 **Statement not reached.**

This statement will never be executed because a statement before it unconditionally transfers control to another statement.

Listing 4.126 Not reaching a statement

```
int foo(int x) {
  return x;
  x++;      // ERROR: This is never reached.
}
```

Error 14132 **Static methods can't be abstract: *method***

You cannot declare a method to be both static and abstract. A static method cannot be overridden. An abstract method must be overridden to be useful.

Error 14120 **Static methods can't be overridden. Method *method* is static in *class*.**

You tried to override *method*, which was declared to be static in *class*.

Fix Either remove the `static` modifier, or remove or rename one of the methods.

Listing 4.127 Overriding a static method.

```
class A {
  static void foo() { /* . . . */ }
}

class B extends A {
  void foo() { /* . . . */ } // ERROR: foo() is static in A.
}
```

Error 14002 **String not terminated at end of input.**

You forgot to end a string with a closing quote.

Fix Use Syntax Coloring (described in the *CodeWarrior IDE User's Manual*) to find out where the quote is missing.

Listing 4.128 String without end

```
public class TrivialExample {
  public static void main(String args[]) {
    String s = "ERROR: This string is not closed.
  }
}
```

Error 14003 **String not terminated at end of line.**

You didn't end a string at the end of a line.

Fix If you have a long string that you must continue over two lines, split it into multiple strings, concatenated with plus signs (+).

Listing 4.129 Extending a string over a couple lines.

```
String s1 = "ERROR: This string is not closed
                    at the end of the line.";
```

```
String s2 = "OK: This is how to handle" +
            "   a long string";
```

Error 14165 **Superclass *class1* of *class2* not found.**

While defining *class2,* you tried to extend *class1* without defining or importing it. Make sure you spelled the name correctly, defined it in the right place, or imported the class's package correctly.

Listing 4.130 Spelling a superclass name wrong

```
class Fubar { /* . . . */ }
class X extends Foobar { /* . . . */ }
  // ERROR: "Foobar" should be "Fubar"
```

Error 14143 **Superclass of *class* can't be an interface: *interface***

You declared a class *class* and used the `extends` keyword where you should have used the `implements` keyword.

Fix When you list the interfaces of an class , use the `implements` keyword.

Listing 4.131 Using extends, instead of implements

```
interface CanFly { /* . . . */ }

class Plane extends CanFly { /* . . . */ }
  // ERROR: Use implements, not extends

class Bird implements CanFly { /* . . . */ }
  // OK
```

Error 14247 **The class file containing *aClassName* has a malformed *AttributeName* attribute.**

If a class loaded in binary form from a `.class`, `.jar`, or `.zip` file has a bad attribute, this error is generated.

Error 14180 **The class name *className* is already defined in this scope. An inner class may not have the same simple name as any of its enclosing classes.**

You cannot declare duplicate class names. An example of this error is shown in Listing 4.132.

Listing 4.132 Class name defined twice in the same scope

```
public class feem {

  class bar {

    class feem { //error
    }
  }
}
```

Error 14173 **The compiler failed to compile the field *<field name>* in class *<class name>* due to *<internal error message>*.**

This error means that you have found an internal compiler error. Please submit a bug report, preferably via email to <support@metrowerks.com>. Be sure to include code that triggers this error so that our engineers can resolve this compiler bug.

Error 14230 **The definitions of method *method* inherited from *location* and *location* are compatible, but the combination of them is nontrivial and has not been implemented. As a workaround, declare *method* explicitly in this class.**

This error is given when a class `C`

- inherits a definition of method `M` from two sources (for example, a superclass and an interface, or two interfaces), but does not itself declare `M`, and
- neither of the two inherited declarations of `M` are more specific than the other.

For example:

```
public interface One {

  public void aMethod(int i) throws IOException;
}

public interface Two {

  public void aMethod(int i) throws ClassNotFoundException;
}


public abstract class aClass implements One,Two {

}
```

Since aClass does not implement aMethod itself, and since neither method is more specific than the other, it

is impossible (well, very hard) for the compiler to pick one for the definition of aClass. To fix this, you can declare one or the other in aClass.

Error 14222 **The instance method *method* declared in *location* cannot override the static method of the same signature declared in *location*. It is illegal to override a static method.**

You cannot override (or intentionally hide) a static method with an instance method. For example:

```
public class baseClass {

  public static void aMethod() { }
}

public class subClass extends baseClass {

  public void aMethod() { } //illegal
```

```
}
```

Error 14225 **The method *method* declared in *method* (which is not deprecated) overrides a deprecated method of the same signature declared in *method*.**

This error is only issued when the option **Warn about deprecated methods** is enabled in the Java Language settings panel. This errors warns you that a class is overriding a deprecated method, without declaring the override as deprecated. For example:

```
public class baseClass {

  /**
   * @deprecated
   */
  public void aMethod() { }
}

public class subClass extends baseClass {

  public void aMethod() { } //warning here

}
```

Error 14224 **The method *method* declared in *location* cannot override the final method of the same signature declared in *location*. Final methods cannot be overridden.**

You tried to override a method *`method`,* but it was declared to be final in the class *`class`*. It is illegal to override a final method.

For example:

```
public class baseClass {

  public final void aMethod() { }
}
```

```
public class subClass extends baseClass {

  public void aMethod() { } //illegal


}
```

Error 14226 **The method *method* declared in *location* cannot override the method of the same signature declared in *location*. The access modifier is made more restrictive.**

This error occured because a subclass may not hide a method by overriding it with a more restrictive access modifier.

```
public class baseClass {

  public void aMethod() { }
}

public class subClass extends baseClass {

  private void aMethod() { } //error here

}
```

Error 14227 **The method *method* declared in *location* cannot override the method of the same signature declared in *location*. They must have the same return type.**

This error occurred because overriden methods must have the exact same return signature.

```
public class baseClass {

  public String aMethod() { }
}

public class subClass extends baseClass {

  private Object aMethod() { } //error here
```

```
}
```

Error 14228 **The method *method* declared in *location* cannot override the method of the same signature declared in *location*. Their throws clauses are incompatible.**

When overriding a method, all declared exceptions must either be explicitly declared in the throws clause of the overriden method, or the overriden method must contain a superclass of the declared exception.

For example:

```
public class baseClass {

  public String aMethod() throws IOException { }
}

public class subClass extends baseClass {

  private Object aMethod() throws ClassNotFoundException{ }
//error here

}
```

See the Java Language Specification (JLS 8.4.4) for further details.

Error 14229 **The method *method* inherited from *location* is incompatible with the method of the same signature inherited from *location*. They must have the same return type.**

This error is given when a class `C`

- inherits a definition of method `M` from two sources (for example, a superclass and an interface, or two interfaces), but does not itself declare `M`, and
- the two inherited declarations of `M` have a different return type.

For example:

```
public class BaseClass {

  public void aMethod() { }
}

public interface anInterface {

  public int aMethod();
}

public class Subclass extends BaseClass implements anInterface {

  //error issued when checking this Subclass, because the
inherited
  //definitions of aMethod conflict in return type.
}
```

Error 14223 **The static method *method* declared in *location* cannot hide the instance method of the same signature declared in *location*. It is illegal to hide an instance method.**

One cannot override (or intentionally hide) an instance method with a static method. For example:

```
public class baseClass {

  public void aMethod() { }
}

public class subClass extends baseClass {

  public static void aMethod() { } //illegal


}
```

Error 14248 **The type *className* to which the member *memberTypeName* belongs is not accessible from *AccessingClassName*.**

If an attempt is made to reference a static field, or the length of a static array field, and the containing class of the given field is not accesible from the class being compiled, this error is generated. For example:

```
class A { private static class C {  public static int i; } }
class B { static void method() { int j = A.C.i; }} //error
```

Note that the field `i` is public, but the inner class `C` is private to `A`, and thus inaccesible from `B`.

Error 14185 **The type *typeName* can't be private. Package members are always accessible within the current package.**

Top level classes (i.e. non-nested classes and static nested classes) cannot be private.

Listing 4.133 Declaring a class to be private

```
private class A { /* . . . */ } // ERROR
class B { /* . . . */ } // OK: Only available to classes
                        // in this package.
public class C { /* . . . */ }// OK: Available to all classes
                              // in any package.
```

Error 14210 **The type *innerClassName* can't be static. Static members can only occur in interfaces and top-level classes.**

Non-top-level classes can't contain static inner classes.

```
public class outerClass {

  public class innerClass {

    static class innerInnerClass {
      //illegal; innerClass isn't top level
      //(i.e. isn't package level and isn't static)
```

```
    }
  }
}
```

Error 14186 **The type *typeName* can't be declared static. It is already top-level, since it is a member of a package.**

Package-level classes shouldn't be declared static, as they are implicitly static.

```
public static class topLevel {
//if this class is not nested, remove the static qualifier
}
```

Error 14187 **The type *typeName* can't be made protected. Package members can either be public or local to the current package.**

Non-nested classes can either be declared "public" or left unqualified. Protected package-level classes are not allowed.

```
protected class topLevel {
//if this class is not nested, remove the protected qualifier
}
```

Error 14232 **The variable in an assignment to a blank final must be a simple name or a simple name qualified by "this".**

In order to determine if blank final instance variables are definitely assigned or unassigned, the compiler requires that their assignment be non-qualified or qualified with "this". For instance:

```
public class aClass {

  public final int aBlankFinal;

   {
    aClass.aBlankFinal = 1; //illegal; don't qualify
  }
}
```

Error 14231 **The variable in an assignment to a static blank final must be a simple name (it may not follow a dot '.').**

In order to determine if static blank final variables are definitely assigned or unassigned, the compiler requires that their assignment be non-qualified. For example:

```
public class aClass {

  public static final int aBlankFinal;

  static {
    aClass.aBlankFinal = 1; //illegal; don't qualify
  }
}
```

Error 14138 **This final variable must be initialized: *symbol***

Final variables in interfaces must be initialized upon declaration, as interfaces can't have code that would initialize a blank final.

For example:

```
interface A {

  static final int i;     //illegal
  static final int i = 3; //ok
}
```

Error 14136 **Transient variables can’t be members of interfaces: *symbol***

Variables in interfaces are by definition public static final, and therefore cannot be transient (because "transient static" would be meaningless, as static variables aren't serialized.)

Error 14023 **“try” without “catch” or “finally”.**

A `try` statement must be followed by at least one `catch` or a `finally` clause. This `try` statement has none.

Fix Make sure your braces are balanced correctly.

Error 14015 **Type expected.**

This error will be signaled when the compiler expects a class name or a primitive type, but finds something else. This occurs in various situations. An example of this error is below in Listing 4.134

Listing 4.134 type expected

```
public static final name = "hi";  //should be "public static final
                                   //String name = "hi";
```

Error 14184 **Type *typeName* is inherited in *className*, and hides a type of the same name in an enclosing scope. An explicit qualifier prefix must be used to name this type.**

An example of this error and its fix are shown in Listing 4.135.

Listing 4.135 Failure to name a type

```
public class bar {

  public class foo {

  }

}

public class feem {

  public class foo {

  }

  public void aMethod() {

    bar aBarSubclass = new bar() {

      public void innerMethod() {
```

```
        foo aFoo = new foo(); //if bar.foo is meant to be
                              //instantiated,
            //"foo afoo = new bar.foo();" should be used
      }
    }
  }
}
```

Error 14005 **Unbalanced parentheses.**

You left out a parenthesis in an expression.

Fix Use the Balance command (described in the *CodeWarrior IDE User's Manual*) to find out where the parenthesis is missing.

Listing 4.136 Missing a parenthesis.

```
System.out.println("Hello World!" ;// ERROR
```

undef.var.or.class

See "Undefined variable or class name: className.typeName" on page 268.

Error 14053 **Undefined variable: *variable***

You used *variable* without defining or importing it.

Fix Make sure you spelled the name correctly, defined it in the right place, or accessed it through a class instance if necessary.

Listing 4.137 Using an undefined variable

```
class A {
  int foobar = 0;
}

public class TrivialExample {
  public static void main(String args[]) {
```

```
    int fubar = 0;
    A a = new A();

    int x = foobar + 1;// ERROR
    int y = a.foobar + 1;// OK
    int z = fubar + 1; // ERROR
  }
}
```

Error 14054 **Undefined variable: *variableName*. The "super" keyword may only be used for member access and constructor invocation.**

If the super keyword is used to qualify something other than a field or method of a super class, this error will result.

Listing 4.138 Using the super keyword to qualify something other than a field or method of a superclass

```
public class one {

}

public class two {

  public two() {
    super.blort(); // ERROR
  }
}
```

Error 14056 **Undefined variable, class, or package name: *a.qualified.Name***

If a qualifed name is used, and the qualifier cannot be resolved to a class or a package, this error will result, as shown in Listing 4.139.

Listing 4.139 Unresolved qualifer

```
  String s = java.util.File.fileSeparator;
//should be "java.io.File.fileSeparator"
```

Error 14055 **Undefined variable or package name: *a.qualified.Name***

If a qualifed name is used, and the qualifier cannot be resolved to a package, this error will result.

Error 14057 **Undefined variable or class name: *className.typeName***

If a qualifed name is used to reference a member of an object, and the qualifier cannot be resolved to a class or variable, this error will result.

Listing 4.140 Undefined variable or class name

```
Thred.currentThread(); //Should be "Thread.currentThread();
```

Error 14258 **Unexpected end of file encountered while reading the class file *fileName***

When reading from a `.class`, `.zip`, or `.jar` file, if the class in question is shorter than expected, this error is generated.

V to Z (Java)

These are error messages that begin with *V*, *W*, *X*, *Y*, or *Z*.

Error 14211 **Variable *variableName* can't be static in *innerClassName*. Only members of interfaces and top-level classes can be static.**

Non-top-level classes can't contain static variables.

```
public class outerClass {


  public class innerClass {

    static int x;
       //illegal; innerClass isn't top level
       //(i.e. isn't package level and isn't static)
```

```
  }
}
```

Error 14041 **Variable *variable* in *location* not accessible from *class*.**

You tried to access *variable* in *location* , but its access declared in *class1* won't allow it.

Fix Either change *variable*'s access status or rewrite your code so you don't need it.

```
class A {
  private int x = 1;
  protected int y = 1;
}

class B extends A {
  int a = x;// ERROR: x is private.
  int b = y;// OK: y is protected.
}
```

Error 14181 **Variable *variableName* is inherited in *className*, and hides a variable of the same name in *anscestorClassName*. An explicit "this" qualifier must be used to select the desired instance.**

An example of this error and its fix are shown in Listing 4.141.

Listing 4.141 Failure to increment a variable

```
public class bar {

  public int foo;

}

public class feem {

  public int foo;

  public void aMethod() {
```

```
    bar aBarSubclass = new bar() {

      public void innerMethod() {

        foo++; //if (e.g.) feem.foo is meant to be
               //incremented, feem.this.foo++ should be used
      }
  }
  }
}
```

Error 14182 **Variable *variableName* is inherited in *className*, and hides a local variable of the same name. An explicit "this" qualifier must be used to select the variable, or the local must be renamed.**

An example of this error and its fix are shown in Listing 4.142.

Listing 4.142 Failure to increment a local variable

```
public class bar {

  public int foo;

}

public class feem {

  public void aMethod() {

    int foo;

    bar aBarSubclass = new bar() {

      public void innerMethod() {

        foo++; //if feem.foo is meant to be incremented,
               //feem.this.foo++ should be used
      }
```

```
    }
  }
}
```

Error 14058 **Variable *variable* may not have been initialized.**

You're using *variable*'s value, but you haven't initialized it yet.

Fix Either initialize the variable or rewrite your code so you don't need it.

Listing 4.143 Using an uninitialized variable

```
int x;
int y = x+1; // ERROR

x = 2;
int z = x+1; // OK
```

Error 14178 **Variable *symbol* is already defined in this method.**

You declared *variable* twice in this method.

Fix Either remove one of the declarations, or rename one of them.

Listing 4.144 Using the same variable name twice

```
int x;
// . . .

for (int x = 1; x<10; x++) { // ERROR: x already defined
  //  . . .
}
```

Error 14177 **Variable *variable i*s used twice in the argument list of this method.**

You declared *variable* twice in this argument.

Fix Either remove one of the declarations, or rename one of them.

Listing 4.145 Defining an argument twice

```
int foo(int x, int y, int x) { return x+y+x; }
  // ERROR: x defined twice
```

Error 14255 **Variables can't be declared strict or wide: *variableName***

The compiler does not allow you to declare variables strict or wide. For example:

```
 class A {
  strictfp float i;
 }
```

Error 14134 **Variables can’t be synchronized, abstract or native: *variable***

You cannot declare a variable to be synchronized, abstract, or native. These modifiers describe how a method is implemented and don’t make sense when applied to variables.

Error 14137 **Volatile variables can’t be final or members of interfaces: *variableName***

You cannot declare a volatile variable as final, or as a member of a certain interface.

Listing 4.146 Declaring a volatile variable as final

```
public class aClass {

  public volatile final int foo; //error
}
```

Error 14096 **Wrong number of arguments in *methodName*.**

When the compiler can determine which method you are trying to invoke, but also determines that you have used the wrong number of arguments to that method, this error will be displayed.

5

Linker Error Messages

This chapter gives an alphabetical list of the most common linker errors which may be encountered while using Metrowerks CodeWarrior compilers.

In this chapter, errors with variable initial text (such as a class or function name) come first. Errors beginning with a non-alphabetic symbol character come next. After that, errors are listed alphabetically.

Linker Errors

The linker errors are divided into the following sections:

- Symbol Names (Linker)
- A to C (Linker)
- D to F (Linker)
- G to I (Linker)
- J to L (Linker)
- M to O (Linker)
- P to T (Linker)
- U to Z (Linker)

Symbol Names (Linker)

These are linker error messages that begin with a symbol name, or the name of a variable or function.

_foo 16-bit code reference to *_bar* is out of range.

The target symbol of a 16-bit PC-relative jump must be located within 32K of the jump statement.

Fix There are two possible fixes to this error message.

First, rearrange the files in your project to move the target closer to the source.

If this doesn't solve the error (and you are targeting Mac68k), set the Code Model setting to large. To do this, open the **Settings Panel Dialog** from the **Edit** menu, and choose the panel 68K Processor, under the heading **Code Generation**. Choose Large from the Code Model pop-up menu.

_filename: symbol1 16-bit data reference to *symbol2* is out of range

Symbol1, in the file *_filename*, assumes a data object, *symbol2*, is within a 32K range (a near, 16-bit reference), but *symbol2* is more than 32K away from *symbol1*.

Either *symbol2* must be made closer to *symbol1* to allow a 16-bit reference or *symbol1* must use a far, 32-bit reference to *symbol2*.

Fix Fixing this linker error may require explicitly placing *symbol1* and *symbol2* in the same segment. In the Processor settings panel, try setting the Code Model to Large, selecting the Far Data and Far Strings checkboxes.

_foo 8-bit (16-bit) computed reference out of range: *bar-fa*

A computed reference cannot be resolved because the objects are too far away from each other. This will most likely come from a converted MPW library.

func* has 16-bit code reference to non-code symbol: *s_name

A non Metrowerks assembly code which references a data object using a 16-bit displacement, and the data object is more than 32K away.

Fix There are two ways to fix this. First, move the data object closer to the code which references it (either by using one of the code sorting options in the PPC PEF panel, or by moving files in the project window). The second way to fix this is to use an instruction with an absolute address or a bigger displacement.

***func1* 16-bit code reference to *func2* is out of range**

A non Metrowerks written assembly code which references a second function using a 16-bit displacement, and the data object is more than 32K away.

Fix There are two ways to fix this. First, move the function closer to the code which references it (either by using one of the code sorting options in the PPC PEF panel, or by moving files in the project window). The second way to fix this is to use an instruction with an absolute address or a bigger displacement

_*foo* has 16-bit data reference to non-data symbol *bar*

bar is referenced as data but defined as code.

_*foo* has illegal computed reference between segments: *bar-fa*

There is an illegal reference type in one of the object files. This will most likely be caused by a converted MPW library.

_*foo* has illegal single segment 16-bit reference to *bar*

This error message is given when you try to use more than one segment in a non-extended resource.

Fix To fix this linker error get rid of the segmentation or to switch to an extended resource by selecting the **Extended Resource** checkbox in the 68K Project preferences.

See Also: For information on code models and code resources, see Targeting Mac OS.

_*foo* has illegal single segment 32-bit reference to *bar.*

Single segments cannot use multiple segments or jump tables. This error message is displayed when some code was compiled with smart or large code model and the project type doesn't support it.

NOTE: Often, the problem is from using an ANSI Library since the ANSI library was built with smart code model. If you want to use an ANSI Library, you have to make a multi-segment code resource by setting the extended resource option in your 68k project preference.

Fix Smart or large code can only be used in multi-segment projects, like applications and multi-segment code resource. It can't be used with single-segment resources. If you want to make a multi-segment code resource, turn on the **Extended Resource** option in the 68K Project settings panel

See Also For information on code models and code resources, see *Targeting Mac OS*.

_*foo* is undefined or is not an exportable object.

This error message is given if the user has an entry in the `.exp` file, and the entry cannot be found or cannot be exported (i.e. marked as `#pragma internal`).

_*filename* is not a valid library file.

The named file had the proper file type for a CodeWarrior library, but the contents are not valid. The library file in question may be damaged.

_*filename* is not a valid PEF shared library.

The named file had the proper file type for a shared library, but the contents are not valid. The shared library in question may be damaged.

_*filename* is not a valid XCOFF file.

The named file had the proper file type for an XCOFF library or shared library, but the contents are not valid. The XCOFF file in question may be damaged.

_*foo* referenced from _*bar* is undefined.

The Linker will generate this error message because a referenced code or data module is not defined anywhere. This means that the undefined function does not exist in your code. This could be because it is a library that you did not include, a source file you did not include or because you forgot to define the function in your code.

Fix Locate or create the referenced function and make sure it is in your project.

A to C (Linker)

These are linker error messages that begin with *A*, *B*, or *C*.

A *<c><num>* resource was found. It will override some project settings.

(Warning) This linker warning is generated when <c><num> resource was found that will override some project settings.

Fix This is common when creating a FAT project. The linker creates 'SIZE' and 'cfrg' resources based on the project preferences. If the linker finds a user-defined resource of either one of these types and ID's, then it uses that resource instead of its own. The warning tells you not to expect certain project preferences to work.

An error occurred while importing the shared library.

This general error message is generated by the PEF Importer to flag a read error.

An error occurred while reading from the .exp file.

This general error message is generated PEF Linker is to flag a error while reading the `.exp` file.

An error occurred while linking the PEF container.

This is a general purpose error message given by the PEF linker for CFM68K. It is used to flag write errors.

An error occurred while trying to open the .exp file.

The linker generates this general purpose error message if reading of the `.exp` file fails.

An error occurred while trying to write the .exp file.

This general error message is generated PEF Linker is to flag a error while writing the `.exp` file.

application has no main entry point

An entry point has not been entered in the Linker settings panel.

application or code resource has no main entry point.

An application or code resource must have a main entry point specified in the PEF settings panel. If you attempt to link a PowerPC project that does not have a main entry point, this link error occurs. For more on entry points and the PEF preferences, consult "Setting the PEF Preferences (PowerPC Only)" in *Targeting Mac OS*.

Bad relocation symbol: *ErrNum*

This is an internal linker error indicating a bad object.

Cannot include more than 1 resource file

Your project has tried to include more than one resource file.

Cannot load object resource for *bar*

An object resource cannot be loaded. This is probably caused by either a corrupt project file or a memory problem.

Fix Delete object file and preference file and check your hard drive and memory for possible errors with a disk utility.

Cannot use CFM68K library.

This error message is generated when you try to use a CFM68k library in a traditional Mac OS application or resource

Cannot use non-CFM68K library.

This error message is generated when you try to use a a traditional Mac OS library in CFM68k application or resource.

Can't copy resource file *filename*.

An I/O error occurred while copying one of the resource files into the final executable file. The specific I/O error is indicated as well.

Can't find source file '*foo*' — statement locations will be lost.

(Warning) The linker could not find the file named *foo* while importing XCOFF object code. When you debug the code, the debugger may not be able to display the source code for the file, or the debugger may display the source code but won't be able to display dashes for breakpoints.

Can't import PEF shared library *filename.*

An I/O error occurred while reading the named shared library file during Make. The specific I/O error is indicated as well.

Can't import XCOFF file *filename.*

An I/O error occurred while reading the named XCOFF file during Make. The specific I/O error is indicated as well.

Can't parse debug information in '*string*'

(Warning) The linker could not understand a piece of debug information while importing XCOFF code. The debugger may not display the symbol associated with the string correctly.

Can't read export file *filename*.

An I/O error occurred while reading the `.exp` file for this project. The specific I/O error is indicated as well.

Can't read library file *filename*.

An I/O error occurred while reading the named library file during linking. The specific I/O error is indicated as well.

Can't read sort file *filename*.

The linker was unable to open and read *filename*.

Fix The file may be corrupted or other input/output error. You should use a disk utility to check your hard drive for possible flaws. If no flaws are found please contact Metrowerks Technical Support.

Can't read temp file '*filename*'

(MPW) You use the `-tmp` directive, and the linker cannot read from or write to one of the temporary files that the linker created in the folder you specified.

Fix Make sure the name of the folder is correctly spelled, the path is correct, the folder's volume is mounted, and that you have access privileges to it.

Can't write application *filename.*

An error occurred while the linker was performing file access on the executable file (for example, reading, writing, or opening the file). This executable file can be an application, shared library, or code resource. The specific error is indicated as well. It can be an I/O error such as "Disk Full" or it can be an Out of memory error.

Fix Memory errors can usually be fixed by increasing the partition size of CodeWarrior.

Can't write export file *filename*.

An I/O error occurred while the linker was writing the `.exp` file. The specific error is indicated as well. If you select "Use .exp file" from the PEF settings panel of the Preferences, but the file *project-name*`.exp` does not exist, the linker will write a default `.exp` file containing all global symbols.

See Also "Export Symbols" in *Targeting Mac OS*.

Can't write library file *filename*.

An error occurred while the linker was writing the library file. The specific error is indicated as well. This can be an I/O error such as "Disk Full" or it can be an Out of memory error.

Fix Out of memory errors can usually be fixed by increasing the partition size of CodeWarrior.

Can't write link map *filename.*

An error occurred while the linker was performing file access on the link map file (for example, reading, writing, or opening the file). The specific error is indicated as well. This can be an I/O error such as "Disk Full" or it can be an Out of memory error.

Fix Out of memory errors can usually be fixed by increasing the partition size of CodeWarrior.

Can't write sort file *filename*.

The linker failed to write the sort file *filename*.

Fix The file may already be opened or corrupted, or other possible problems: disk full, file is locked, i/o error, disk is write-protected. If other conditions occured please contact Metrowerks Technical Support.

Can't write SYM file *filename.*

An error occurred while the linker was writing the SYM file. The specific error is indicated as well. This can be an I/O error such as "Disk Full" or it can be an Out of memoryerror.

Fix Out of memory errors can usually be fixed by increasing the partition size of CodeWarrior.

Class optimization failure for *function*

The linker failed to optimize the function *function*.

Fix The unit declaring *function* class wasn'tcompiled using 'optimize class hierarchy'.

code resource must not have a termination routine

A native or fat code resource cannot have a termination entry point, specified in the PEF settings panel.

COMDAT sections do not match in size: *Func*

Multiple identical definitions for functions are allowed but these do not match in size. This is not currently supported.

cross-TOC call from *symbol1* to *symbol2* has no TOC reload slot

The linker requires that a `NOP` instruction be placed after any call to an external routine, as a place-holder for a TOC-reload instruction needed when calling a routine from a shared library. This error can occur if you import some assembly language code that does not

contain the `NOP`, but calls a routine which is imported from a shared library.

D to F (Linker)

These are linker error messages that begin with *D*, *E*, or *F*.

Duplicate COMDAT section: *Func* in files:

Invalid multiple definition for functions.

entry-point *symbol* is not a descriptor

(Warning) The initialization, main, and termination entry points are usually external routines, that the linker references through a descriptor (sometimes called a TVector) in the data area. For certain kinds of code resources (like plugins), you may want to use a variable as the entry-point instead of a function.

Entry Point *_pt* is undefined.

In the CFM68K settings panel, the user can specify three entry points: Initialization, Main, and Termination. This message occurs when you specify an entry point that doesn't exist.

Error creating output file: *filename status*

The linker was not able to create an output file *filename* with the DOS status.

Fix Check to see if there already is an open file with the same filename that is currently in use. Check for free disk space.

Error opening file: *filename status*

The linker could not open *filename* with the DOS *status* for linking.

Fix Reset access paths and verify folders in which the file is located.

Error processing : command file *command file name* : address *address* not a valid address.

You have entered a non-32 bit address, or an address that is not in hex format (`0xnnnnnnnn`). This error could also occur if you've neglected to enter an address when one is expected.

Error processing : command file *command file name* : expected *symbol* found *symbol*.

This error indicates a linker command file syntax error.

Error processing: command file *command file name* : too many left braces.

The number of left braces is not balanced with the number of right braces in the linker command file.

Error processing: command file *command file name* : too many right braces.

The number of right braces is not balanced with the number of left braces in the linker command file.

Error processing : command file *command file name* : unexpected end of file.

The linker expected more data when processing the command file. This error indicates a probable linker command file syntax error.

For example, when the command file parser finds an open brace, then some section names, and then finds the end of the file before the closing brace is found, this error is generated.

Error processing : command file *command file name* : unknown keyword *keyword*.

You've used a string in the keyword position that the linker command file parser does not define.

Error section mapping: command file *command file name* is not text.

The linker command file used in your project is not a text file.

Error section mappings: section *section name* assigned more than once.

The section name has already been used in the same or different segments.

Error section mappings: segment *segment name* : contains no sections

This error indicates that a segment listed in the Section Mappings settings panel or the linker command file does not have any sections associated with it (i.e. none of the sections listed for the segment are used in the output file.)

Error while *operation*

Linker errors in this form occur when the linker encounters file I/O problems. *Operation* may one of the following:

- saving resources
- writing to SYM file
- setting SYM file position
- getting file info
- creating SYM file
- opening SYM file
- reading or parsing SEG file
- creating new file
- creating or copying resource fork
- opening file's resource fork
- writing file's resource fork

These linker errors can be caused by anything from defective media to a volume being full. The solution depends on the nature of the error.

Error writing output file: *filename*

The linker was unable to write to the output file *filename*.

Fix This is likely due to lack of drive space. Also, check to see if the Drive is corrupted.

export symbol *symbol* is undefined

The symbol named in the `.exp` file does not exist.

File read error

The linker could not read the file for some reason.

Fix verify that the file is present and is not corrupted.

G to I (Linker)

These are linker error messages that begin with *G*, *H*, or *I*.

Global Object *obj* was already declared in File: *fname.*

The linker will generate this error if you define a global data or code object in more that one file.

Fix Remove all but one declaration, or declare all but one to be external.

HUNK_DEINIT_CODE not yet supported.

This error message is generated when an unsupported object type is encountered.

NOTE: You will not get this error unless your object file has been corrupted.

---

ignored duplicate resource *type* (*id*) in *filename*

The named resource was already copied from another resource file in the project.

ignored: *symbol class* in *filename1* previously defined in *filename2*

(Warning) The linker will permit multiple definitions of a symbol that is defined in a library. The order of files in the project window determines which definition of the symbol will be used. Subsequent definitions are ignored.

Illegal computed reference for *foo*

The linker encountered an illegal computed reference for *foo*.

Illegal relocation for *foo*

An illegal relocation information.

Illegal object data

This error message is generated when the linker encounters illegal data in a library.

Illegal object data in *objectFile*

The linker does not recognize the data format in file *objectFile*. Usually, this error is issued when you attempt to link a PowerPC library with a 68K project.

Fix Make sure the file is in a format recognized by the CodeWarrior project manager. To fix this error, build the library's source code with a 68K compiler and linker or find a 68K version of the PowerPC library.

Illegal object file version, an error occurred while writing the library's object data.

An error occurred during the file I/O.

Incorrect ELF header - e_ident[*ELF Identification*] in file *filename subfilename*

There is an invalid `e_ident` entry in the ELF header.

Internal error *ErrMessage*

An internal linker error has occurred with the Metrowerks linker. Please report this to Metrowerks Technical Support.

internal linker error: File: *filename* Line: *line number*

An internal linker error has occurred with the Metrowerks linker. Please report this to Metrowerks Technical Support.

Invalid object code.

You tried to link a 68K library with a PowerPC program.

Fix To fix this error, build the library's source code with a PowerPC compiler and linker or find a PowerPC version of the 68K library.

Invalid object file: *filename*

The object file *filename* was of the wrong object format.

Invalid object library

The object Library was an invalid type for the current project type.

Invalid object library: *filename*

The object library *filename* was invalid.

Invalid PEF shared library.

The named file had the proper file type for a CodeWarrior library, but the contents are not valid. The library file in question may be damaged.

Invalid relocation type: *ErrNum*

This is an internal linker error indicating a bad object.

Invalid subsystem

The subsystem id that was set on the linker panel is invalid.

J to L (Linker)

These are linker error messages that begin with *J, K,* or *L.*

Library Linker: Cannot have resource files in library.

This library linker error message is shown if you have a `.rsrc` file in your library project.

NOTE: This error is often generated by including a ResEdit file in the project.

Library Linker: Cannot load object resource.

The library linker generates this message when there is not enough memory. Alternatively the resource file maybe damaged.

Library Linker: Illegal object data.

This error is generated when the linker encounters illegal data in a library.

Library Linker: Out of memory.

This library linker error message is shown when there is not enough memory.

library must not contain any resource files

You can't add any resource files in a project whose type is library. If the library requires resources, the resource files must be added to the projects which use the library.

Library resource cannot be read.

This error message is displayed whenever you have a library file error.

Link aborted - Too many errors.

The linker will generate this error message more than 100 linker errors are encountered.

Link Error: Code resource cannot have more than one segment.

This error appears if you try to have more that one code segment in s single segment resource

Link Error: Object resource not found.

This linker error occurs when an object resource cannot be loaded. This could be a corrupt project file or memory problem.

Link Error: Runtime Object resource not found.

This linker error will not occur unless you have modified the compiler's resources. If you receive this linker error, contact Metrowerks Technical Support.

Link failed.

A fatal link error (or errors) occurred, such as Out of memory . The specific link error is indicated as well.

Local Object *foo* is redeclared

The local object *foo* is declared more than once within one object file.

M to O (Linker)

These are linker error messages that begin with *M*, *N*, or *O*.

'main' is undefined.

This Linker error message occurs when there is no `main()` function defined in the program.

Fix You need to fix this error differently, depending on whether your project is for an application, C code resource or Pascal code resource.

Application In an executable application this error is usually the result of not including the source file that includes the `main()` function in your project.

C code resource In a multi-segment project, your `main()` must be in Segment 1. If your `main()` is not in Segment 1 then the CodeWarrior start-up code tries to call your `main()` via an A4-based reference. However, A4 is not setup until your `main()` has a chance to call `SetCurrentA4()`.

Pascal code resource Code resources can only be built from units. In the units `INTERFACE` section, you have to indicate the code resources main entry point. This entry point is indicated with the `$MAIN` directive. You need to specify the missing `main` identifier with the `$MAIN` directive.

missing vtable: *_vt_foo* Check that all virtual functions and static members are defined

The linker generates this error message because the compiler usually tries to generate only one instance of a virtual function table. This table is usually created together with a certain static member or a virtual function definition. If you forget to define that member you will get this error message.

NOTE: This message is specific to a C++ project. The error usually means you are making an instance of a derived class without declaring the base classes or including the base class' header file in your derived class header files.

Mixed target *message* file *filename* subfile *subfilename*

You have mixed target types in your project.

This error could occur, for example, if you built a library for a NEC V810 target and tried to use it in a NEC V830 project.

Multiple definition of symbol: *Var* in files:

The linker found two or more definitions of the same symbol.

Fix Rename the variables to remove the conflicts.

multiply-defined: *symbol* in *filename1* defined in *filename2*

The linker will not permit multiple definitions of a symbol if neither symbol comes from a library.

NOTE: All multiply-defined symbols are reported in a single message.

Multiply defined symbol *symbol name*. Defined in *filename list*.

The linker found two or more global definitions of the same symbol.

Near data section or jump table is greater than 32KB.

With CFM68K the near data section uses A5-relative addressing, so is limited to 32KB below A5. The jump table is above A5 and also limited to 32KB.

Near data segment is bigger than 64k

The project being linked has more than the 64K limit of global data allowed using near, 16-bit references. To allow more than 64K of global data, your project needs to use far, 32-bit references.

Fix Select the **Far Data** and **Far Strings** checkboxes in the 68K Processor settings panel.

No entry point found

The linker did not find an entry point.

Fix Check to see if your project includes a main or WinMain function. Check to see if your Project settings x86 Linker options have the entry point correctly listed.

Not a *CPU_Type* Library

This Library importer encountered an incorrect library type for the project target.

Fix Replace the offending library with the correct CPU version.

Not enough memory for linker.

This error message is given if there is not enough memory for the linker.

Out of memory

The linker does not have enough memory to complete the build.

Fix Increase the partition for Macintosh-hosted compilers, or increase the system memory for Macintosh and Windows hosted compilers by closing other applications, or increasing the memory.

P to T (Linker)

These are linker error messages that begin with *P*, *Q*, *R*, *S*, or *T*.

sort symbol *s_sym* is undefined

(Warning) A sort symbol `s_sym` was undefined.

Fix The sort file names a symbol that the linker didn't find. Either define that symbol, or remove it from the sort file.

sort file *FileName* did not list all code symbols

(Warning) This linker warning is given when the file `FileName` failed to list all possible code symbols.

Fix It's not really necessary to fix this unless it concerns you. The code will be in the order specified in the sort file until the sort file runs out of symbols. The rest of the code symbols follow, in an undefined order.

Sorting *obj_name*

This is not an error it is a status message, like "Linking" or "Compiling".

Symbol *symbol name* in *filename* too far for a reference from section *section name* in *filename*, jump island created.

If a jump to a target address requires more than 26 bits, a jump island is created and you get this message as a warning. If the distance is still too large, you will get this message as an error.

Symbol data error in *bar*

The linker found illegal data in the object file *bar*. This means that the object data is not compatible with the current CodeWarrior compiler. Remove all binary information and recompile your project.

Syntax error in exports file *fname* , line *n*.

The export file must contain only identifiers, comments (indicated by a # character and terminated by the end of line) and white space (spaces or tabs)

syntax error on line *n* of export file *filename*

The export file must contain only identifiers, comments (indicated by a # character and terminated by the end of line) and white space (spaces or tabs).

syntax error on line *LineNo* of sort file *FileName*

The sort file had a syntax error on line number `LineNo` in file `File-Name`

Fix Fix the syntax error. The syntax of the sort file is the same as the syntax for an exp file.

The following sections were not assigned to a segment :

A section that makes up part of the output is not assigned to a segment in the Section Mappings settings panel or the linker command file.

The global data module *foo* is undefined

Where `foo` is one of the following internal start-up code variables:

- `__codereftype__`
- `__maincodexnum__`
- `__headersize__`
- `__LoadSeg__`
- `__codexreftype__`
- `__Startup__`

This linker error will not occur unless you have modified the compiler's resources. If you receive this linker error, contact Metrowerks Technical Support.

The main entry point for applications must be an executable module.

The CFM68k linker generates this error if the user puts a non-function as the main entry point for an application (which is illegal), for Shared Libraries

NOTE: You can have a data object as the main entry point.

The __segloader routine cannot be found.

The `__segloader()` routine is used to load segments under CFM68K, it is found in the `MWCFM68KRuntime.Lib` library.

The project and the file *filename subfilename* do not have the same Class Identifier. Cannot mix 32 bit and 64 bit code.

There are different `e_ident[EI_CLASS]` entries in file header than indicated for your target's settings. You are attempting to mix 32 and 64 bit code.

The virtual function table *_vt_foo* is undefined, make sure that all static members and virtual functions are defined.

The linker generates this error message because the compiler usually tries to generate only one instance of a virtual function table. This table is usually created together with a certain static member or a virtual function definition. If you forget to define that member you will get this error message

---

NOTE: This message is specific to a C++ project. The error usually means you are making an instance of a derived class without declaring the base classes or including the base class' header file in your derived class header files.

TOC size of *n* bytes exceeds 64K limit

The TOC is a part of the data area used to indirectly address other data. The TOC contains one 4-byte entry for each variable, floating-point constant, or string constant that is referenced in the program.

Fix You can reduce the TOC requirements of your program with a couple of different option:

- The **Pool Strings** option in the C/C++ Language settings panel, or pragma `pool_strings`, pools the string constants from each file into a single data object which needs only 1 TOC entry.

- The **Store Static Data in TOC** option in the PPC Processor settings panel stores small static integer variables and floating-point constants directly in the TOC, instead of allocating space for them elsewhere and storing pointers to them in the TOC. If you have lots of small static variables (under 4 bytes), turn this option on to save TOC space. If you have lots of large floating-point constants (4 to 8 bytes), turn this option off to save TOC space.

too many link errors

The linker stops reporting errors after about 100 errors are encountered.

U to Z (Linker)

These are linker error messages that begin with *U*, *V*, *W*, *X*, *Y*, or *Z*.

Unable to launch the Application.

This linker error is given if a file I/O problem occurs when CodeWarrior performs a **Run**.

Unable to load multi-segment driver header

This linker error will not occur unless you have modified the compilers resources. If you receive this linker error, contact Metrowerks Technical Support.

Undefined *entry-point name*

The entry point symbol is not defined in any of the object files.

undefined: *symbol1 class* referenced from *symbol2* in *filename*

The named symbol was referenced but never defined. As a special case, you'll get an undefined message for the symbol `__procinfo` (data) if you make a Native code resource that is not one of the known resource types: `CDEF`, `MDEF`, `MBDF`, `LDEF`, `WDEF`, `cdev`, `XCMD`, or `XFCN`.

NOTE: Multiple references to the same undefined symbol are reported in a single message.

Fix To avoid this error, you must define a global variable `__procinfo` of type `unsigned long` initialized to the proper `MixedMode` flags placed in the `RoutineDescriptor` that becomes part of the Native code resource. Consult the comments in the `MixedMode.h` for details on `RoutineDescriptor` flags.

Undefined symbol *symbol name*. Referenced from *referencing symbol* in *filename*.

A symbol referenced by another symbol (for example, a function) is not defined.

Undefined symbol: *Var* in file:

The linker was unable to find the symbol *Var*.

Fix Usually this is due to a spelling error, missing library or a source code that was not added to the project.

unsupported XCOFF relocation (*x*, *y*, *z*) in *filename*

The named XCOFF file was valid but contained some relocation information that is not supported by the CodeWarrior linker.

Fix Make sure that the XCOFF file was assembled without symbolic debugging information. If the error still occurs, contact Metrowerks Tech Support.

User requested termination

The build was terminated by the user.

Index

C

Compiler Errors
- C/C++ 68k 9
- C/C++ PPC 9
- Pascal 68k 87
- Pascal PPC 87

Compiler Errors Win32/x86 9

E

Errors
- Compiler C/C++ 9
- Compiler Pascal 87
- Font Explanations 6
- Linker C/C++ 275

Explanations
- Font Conventions 6

L

Linker Errors
- C/C++ Linker Errors 68k 275
- C/C++ Linker Errors PPC 275
- C/C++ Linker Errors Win32/x86 275

CodeWarrior

Error Reference

Credits

writing lead: John Roseborough

other writers: Ron Liechty, Marc Paquette

engineering: Marcel Achim, Mark Anderson, Berardino Barrata, Pascal Cleve, Michael Cordery, Andreas Hommel, Nick Havens, John McEnerney, Clinton Popetz, Cam Vien

frontline warriors: Metrowerks Technical Support

Guide to CodeWarrior Documentation

CodeWarrior documentation is modular, like the underlying tools. There are manuals for the core tools, languages, libraries, and targets. The exact documentation provided with any CodeWarrior product is tailored to the tools included with the product. Your product will not have every manual listed here. However, you will probably have additional manuals (not listed here) for utilities or other software specific to your product.

Core Documentation	
IDE User Guide	How to use the CodeWarrior IDE
Debugger User Guide	How to use the CodeWarrior debugger
CodeWarrior Core Tutorials	Step-by-step introduction to IDE components
Language/Compiler Documentation	
C Compilers Reference	Information on the C/C++ front-end compiler
Pascal Compilers Reference	Information on the Pascal front-end compiler
Error Reference	Comprehensive list of compiler/linker error messages, with many fixes
Pascal Language Reference	The Metrowerks implementation of ANS Pascal
Assembler Guide	Stand-alone assembler syntax
Command-Line Tools Reference	Command-line options for Mac OS and Be compilers
Plugin API Manual	The CodeWarrior plugin compiler/linker API
Library Documentation	
MSL C Reference	Function reference for the Metrowerks ANSI standard C library
MSL C++ Reference	Function reference for the Metrowerks ANSI standard C++ library
Pascal Library Reference	Function reference for the Metrowerks ANS Pascal library
MFC Reference	Reference for the Microsoft Foundation Classes for Win32
Win32 SDK Reference	Microsoft's Reference for the Win32 API
The PowerPlant Book	Introductory guide to the Metrowerks application framework for Mac OS
PowerPlant Advanced Topics	Advanced topics in PowerPlant programming for Mac OS
Targeting Manuals	
Targeting BeOS	How to use CodeWarrior to program for BeOS
Targeting Java VM	How to use CodeWarrior to program for the Java Virtual Machine
Targeting Mac OS	How to use CodeWarrior to program for Mac OS
Targeting MIPS	How to use CodeWarrior to program for MIPS embedded processors
Targeting NEC V810/830	How to use CodeWarrior to program for NEC V810/830 processors
Targeting Net Yaroze	How to use CodeWarrior to program for Net Yaroze game console
Targeting Nucleus	How to use CodeWarrior to program for the Nucleus RTOS
Targeting OS-9	How to use CodeWarrior to program for the OS-9 RTOS
Targeting Palm OS	How to use CodeWarrior to program for Palm OS
Targeting PlayStation OS	How to use CodeWarrior to program for the PlayStation game console
Targeting PowerPC Embedded Systems	How to use CodeWarrior to program for PPC embedded processors
Targeting VxWorks	How to use CodeWarrior to program for the VxWorks RTOS
Targeting Win32	How to use CodeWarrior to program for Windows